

COMPUTER SOFTWARE APPENDIX

The following computer software appendix is being deposited as part of the specification of this patent application under 37 C.F.R. 1.96 as original copies from computer printout and as two sheets per one patent specification page for the ease of the publication office.

00017718-031604
T09120-07-1860

```
*****
* History:
* Version 0.1 - First Package delivery for demo use.
* Version 0.2 - (03-15-96)
* Site ID filtering to catalog
* Delivery support
*
* Steps()
*
*ig into file and added DPCUpdateConfig()
* Version 0.3 - (03-25-96)
* Support for interactivity field in
*
* ... in catalog by adding minimum
* the committing entry.
* Version 0.4 - (03-29-96)
* Indexes to modem configuration and
* 2400 to fix key reception bugs.
* enable agent to continue getting
* full, even if agent shuts down.
* Version 0.5 - (04-09-96)
* Showing up in Cancel Download list
* istics halfway running(doesn't abend)
* licit) files in catalogue
* t/response/confirm code running but
* yet.
* Version 0.6 - (04-10-96)
* ing Get MLID stats
* astrength()
* Version 0.7 - (04-11-96)
*
* igation Editor
* length Meter
* Version 0.8 - (04-12-96)
* astrength to report 0 when not connected
* nting cost(explicit) file download
* View Database Entries option worked
* entire entry and is more usable.
* Version 0.9 - (04-16-96)
```

```
Added Community/
Added Periodic d
Added DPCGetMLID
Broke modem conf
Added SFX_PUSH s
catalog.
Fixed FreeFreeFr
length check bef
Added baud rate
defaulted them t
Also added code
keys until succe
Fixed push files
Got DisplayMLIDS
Recognized cost(
Got explicit req
NOC doesn't send
Finished impleme
Added DPCGetSign
Added Modem Conf
Added Signal Str
Fixed DPCGetSign
Finished impleme
Modified the way
so that it shows
Completed Turbo
```

```
*****
* Internet support. Can't totally test
*
* away is up and running.
* Version 0.10 - (04-24-96)
*
* ck
*
* lse" in IPSEndRoutine after call
* ts. Turbo Internet works 90%.
* Version 0.11 - (04-25-96)
*
* AppendStringField in DLO to include
* instead of NULL.
* Version 0.12 - (05-16-96)
*
* ase 2.
*
* between ATDT and 1-800...
*
* r commas in phone numbers and prefix
*
* ing to accept & and -
*
* Gateway strings to have (ISP) follow it
* played on if -DEBUG is on command line
* Version 1.00 - (06-11-96)
*
* ud rate to Modem Status string
*
* it doesn't stay in during Internet traffic
*
* formation screen
*
* er edit strings to allow alphas
*
* buffer to 4K(default was 1K)
* Version 1.01 - (06-17-96)
*
* iver comm code into driverio.c
*
* ster call to DPC.LAN so that it could
*
* removes which would allow us to prevent
*
* Version 1.02 - (06-25-96)
*
* ch to disable package delivery for debug
* Version 1.04 - (06-25-96)
*
* time to send directly to AIO.
* Version 1.05 - (06-28-96)
*
* umber in menu to decimal
*
* size field to modem configuration
*
* which changed configuration screens
* Version 1.06 - (07-05-96)
*
* the finer fragment routines(yk).
* rol menu.
```

```
yet until IJ gat
Added ARP loopba
Fixed missing "e
to GatherFragmen
Fixed NMSAppend
character sets 1
Completion of Ph
Places prefix in
Added support fo
Allowed init str
Modified IP and
Debug screen dis
Added connect ba
Fixed Tx LED so
Added Adapter In
Bumped AIO write
Broke out LAN dr
Added Agent Regi
call us when it
abends.
Added -NOPD swit
Revised send rou
Changed serial n
Added AIO buffer
Added auto login
Completely rewro
Added Modem Cont
Added GNU condit
```

ionals.

pt timeouts

utines into PPP.C

Added login scri
Broke out PPP ro

#define AGENT_VERSION InxMSG("1.20 ", 173)

/* Global variable used by DPCAGENT.C

/* Resource Tag variables.

struct LoadDefinitionStructure *NLMHandle = 0;

struct ResourceTagStructure *allLocTag = 0;

struct ResourceTagStructure *timerTag = 0;

struct ResourceTagStructure *ASTag = 0;

struct ResourceTagStructure *asynctag = 0;

/* NUT and screen ID variables.

/* NUTInfo

NUT handle

/* main

#if DEBUG_ALL
struct ScreenStruct *DebugScreenID = 0; /* for OutputToScreen

/*

#endif

ScreenHeight = 0;

WORD

*/

/* Screen Height(25)

ScreenWidth = 0;

/* Screen Width(80)

BackgroundPortal = 0;

LONG

/* NUT B

ackground Portal

/* Process and thread variables.

/*

BYTE

**NLMMessageTable = (0);

/* inter to messages

DPCAgentPID = 0;

/* Main Handler thread PID

DPCFilePID = 0;

/* Packet Handler thread PID

DPCModemPID = 0;

/* Modem Handler thread PID

DPCAccessPID = 0;

/* Access thread PID

DPCInetPID = 0;

/* Turbo Internet thread PID

DPCName[] = {TEXTMSG("\x03\"DPC\", 72)};

/*

/* All threads must exit

/*

/*

/*

/*

/*

/*

/*

/*

/*

/*

/*

/*

/*

```
0x9cc9, 0x8d40, 0xbfdb, 0xe552, 0x6daed, 0xc6b6, 0xf9ff, 0xe876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0xf76f, 0x4434, 0x55bd,
0xad4a, 0xbcc3, 0x8b58, 0x9fd1, 0xb6e6, 0xfae7, 0xc87c, 0xd95d,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
0xbddb, 0xac42, 0x9ed9, 0x8f50, 0xfbe7, 0xea66, 0xd8fd, 0xc974,
0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
0xc6c, 0xfcc5, 0xe65e, 0xfcd7, 0x8868, 0x99e1, 0xb7a, 0xbaf3,
0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
0xdec, 0xfcd4, 0xfdd, 0xec56, 0x98e9, 0x8960, 0xbdb, 0xaa72,
0x6306, 0x728f, 0x4014, 0x5122, 0x34ab, 0x0630, 0x17b9,
0xf4e, 0xfec7, 0xc5c, 0xdadd, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xfccf, 0xea46, 0xdcd, 0xc54, 0xb9eb, 0xa9f9, 0xb70, 0x9b70,
0x8408, 0x9581, 0xa71a, 0xb593, 0x4e22c, 0x3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cfe,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5e95, 0x4ff6c, 0x7d7f, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0x12a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x76df, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0xd68d, 0xc704, 0xf59f, 0x685e, 0x1ce1, 0x0d68, 0x3f3, 0x2e7a,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3f3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xbda3, 0x8238, 0x3b1,
0x6b46, 0x7acf, 0x4854, 0x59ad, 0x2462, 0x3ceb, 0x0e70, 0x1fef,
0xf78f, 0xe606, 0xd49d, 0xc514, 0x1bab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1eef1, 0x0ef78
};
```

/* MUID Statistic globals */

int GenericDescriptionTable[12*8] =

```
{
    InxMSG("Total packets sent:", 103),
    InxMSG("Total packets received:", 141),
    InxMSG("No ECB available count:", 148),
    InxMSG("Send packet too big count:", 156),
    InxMSG("Reserved:", 186),
    InxMSG("Receive packet overflow count:", 214),
    InxMSG("Receive packet too big count:", 215),
    InxMSG("Receive packet too small count:", 216),
    InxMSG("Send packet miscellaneous errors:", 217),
    InxMSG("Receive packet miscellaneous errors:", 218),
    InxMSG("Send packet retry count:", 223),
    InxMSG("Checksum errors:", 290),
    InxMSG("Hardware receive mismatch count:", 291),
    InxMSG("Total send OK byte count low:", 229),
    InxMSG("Total send OK byte count high:", 230),
    InxMSG("Total receive OK byte count low:", 231),
    InxMSG("Total receive OK byte count high:", 232),
    InxMSG("Total group address send count:", 233),
    InxMSG("Total group address receive count:", 251),
    InxMSG("Adapter reset count:", 252),
    InxMSG("Adapter operating time stamp:", 253),
    InxMSG("Adapter queue depth:", 255),
    InxMSG("Send OK single collision count:", 159),
    InxMSG("Send OK multiple collision count:", 256),
    InxMSG("Send OK but deferred:", 264),
    InxMSG("Send abort from late collision:", 265),
    InxMSG("Send abort from excess collision:", 266),
    InxMSG("Send abort from carrier sense", 267),
    InxMSG("Send abort from excessive deferral", 268),
    InxMSG("Receive abort from bad frame alignment", 269)
```

```
);
#define STATS_DATA_WIDTH 13
#define FSD_DATA_WIDTH 40
#define BORDER_WIDTH 3
#define INDENT_WIDTH 3

void ( *BackgroundFuncPtr ) ( LONG portalNumber ) = NULL;
LONG BackgroundPortal;
int GenericLinesStart;
int GblDataCol;
struct DOSCountryInfoStruct GblDOSCountryInfo;

int DebugFlag = FALSE;

/* The array ndeclination contains the Declination in degrees to add or
 * subtract from true azimuth to get magnetic azimuth.
 * The value 0 means that the location is not in mainland US
 */
static float ndeclination[] =
(
    0, 0, 4, 1, 0, 0, 0, -8, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, -3, -4, -6, -8, -10, -11, 12, -13, 0, 0,
    0, 0, 5, 2, -2, -5, -6, -9, -11, -12, -13, -13, -14,
    0, 7, 5, 2, -1, -4, -6, -8, -12, -12, -14, -15, -16,
    0, 12, 9, 6, -1, -4, -6, -9, -11, -12, -14, -16, -17,
    18, 15, 0, 7, 2, -3, -6, -9, -12, -13, -15, -18, -19,
    0, 0, 0, 0, 0, -1, -6, -9, -13, -15, -18, -20, -21, -22
);

#endif LOG_ECB_ACTIVITY
int DPC_TGID;
LogHandle LogClientHandle;
EventHandle LogECBHandle;
#endif /* LOG_ECB_ACTIVITY */

/*
 * Local function prototypes.
 */
void ReturnResources(int sig);
/*
 * ExitHandler(void *handle)
 *
 * Description:
 * This routine is called when the user hits Alt-F10 to exit
 * the utility. We will attempt to verify with the user that
 * they really want to exit.
 *
 * Input:
 * handle - NUT handle
 *
 * Output:
 * nothing
 *
 * Returns:
 * nothing
 */
*****
void ExitHandler(void *handle)
{
    if (NWMSConfirm(InMSG("Exit DSDEBUG?", 88), 0, 0, TRUE, NULL,
        handle, NULL) == TRUE)

```

- 1 if send to cas, 0 if

* cas

* send to hub
* timeout

- inactivity timeout in seconds

* Output:
* nothing* Returns:
* nothing

void SLIPSend(char *pdata, LONG sz, int cas, int timeout)

LROESPkt_t txcas;
LROASPkt_t txhub;

LONG tmlen;

WORD crc;

BYTE *pi, *po;

BYTE slip_data[3072];
int is, os;

if (cas)

/* initialize cas transfer */

CMOVB(pdata, txcas.data, sz);

txcas.length = sz + (txcas.data - ((BYTE *) &txcas));

tmlen = txcas.length;

txcas.length |= 0x8000;

crc = calcrc(INITCRC, (BYTE *) &txcas, tmlen);

txcas.data[sz] = crc & 0xff;

txcas.data[sz+1] = (crc >> 8) & 0xff;

pi = (BYTE *) &txcas;

}

/* initialize hub transfer */

CMOVB(pdata, txhub.data, sz);

txhub.length = sz + (txhub.data - ((BYTE *) &txhub));

txhub.filler1 = txhub.channel = 0;

txhub.control = 0x8000;

tmlen = txhub.length;

crc = calcrc(INITCRC, (BYTE *) &txhub, tmlen);

txhub.data[sz] = crc & 0xff;

txhub.data[sz+1] = (crc >> 8) & 0xff;

pi = (BYTE *) &txhub;

/* Start out with SLIP start character */
po = slip_data;

*po++ = END;

/* Escape any special SLIP characters that may be in the message */
for (is = 0, os = 1; is < (tmlen + 2); is++, os++, pi++, po++){
switch(*pi)

case END:

*po++ = ESC;

*po = ESC_END;

os++;
break;

case ESC:

*po++ = ESC;

*po = ESC_ESC;

os++;

break;
default:
*po = *pi;

break;

/* Finish packet off with SLIP END character */

*po = END;

os++;

/* Send message to the modem thread */
DioSend(slip_data, os, timeout);

EXPORTED FUNCTION

DPCGetBoard(LONG *board,
LONG *controlEntry)

Description:

This routine returns the DPC MLID logical board number
and control entry address to be used to send IOCTL
requests to the DPC mlid. The IOCTL mlid pragma can
then be used to make the request.

Input: board

- Pointer to where to st

ore board number

controlEntry

- Pointer to where to store control entr

y address

Output:

board and controlEntry filled in if successful

Returns:

0 if MLID is active

LONG DPCGetBoard(LONG *board, LONG *controlEntry)

if (DIOBoard == 0)

return(-1);

*board = DIOBoard;

*controlEntry = DIOControlEntry;

return(0);

void
CreateStringWithCommas(LONG number, BYTE *buffer, char *format){
int i;

int j;

int found;

int length;

int commasAdded = 0;

BYTE tmpBuf[128];

BYTE *tmpPtr;

memset(tmpBuf, format, number);

for (i = (length = CStrlen(tmpBuf)), found = 0; i > 0; i--)

```

(
    if ( ( tmpBuf[ i ] >= '0' ) && ( tmpBuf[ i ] <= '9' ) )
    {
        /* we have a digit */
        if ( ++found > 3 )
        {
            found = 1;

            /* shift the string one to the right */
            for ( j = ++length; j > i; j-- )
                tmpBuf[ j ] = tmpBuf[ j - 1 ];

            tmpBuf[ i + 1 ] = GbIDOSCountryInfo.thousandSep[
                commasAdded++;
            ]
        }
    }

    /* Adjust the length of the string back to its original if there are
    ** leading spaces.
    */
    for ( i = 0, tmpPtr = tmpBuf; i < commasAdded; i++ )
    {
        if ( *tmpPtr == ' ' )
            tmpPtr++;
    }
    strcpy( buffer, tmpPtr );
}

void
SecondsToDateAndTime( LONG seconds, BYTE *buff )
{
    char *ascBuf;

    ascBuf = ascTime( localtime( (time_t *) &seconds ) );
    MOVb( ascBuf, buff, 26 );
    buff[ 24 ] = 0;
}

void
FormatElapsedTime( LONG seconds, LONG tenths, BYTE *buff )
{
    LONG    minutes, hours, days;

    /* convert secs to minutes */
    minutes = seconds / 60;
    seconds = seconds % 60;
    hours = minutes / 60;
    minutes = minutes % 60;
    days = hours / 24;
    hours = hours % 24;
    if ( days > 0 )
    {
        NMSprintf
        (
            buff,
            MSG( "%d %s02d%02d%02d%02d%02d", 2937,
                days,
                GbIDOSCountryInfo.timeSep[ 0 ],
                hours,
                GbIDOSCountryInfo.timeSep[ 0 ],
                minutes,
                GbIDOSCountryInfo.timeSep[ 0 ],
                seconds,
                GbIDOSCountryInfo.decimalSep[ 0 ],
                tenths
            );
        }
    }
    else if ( minutes > 0 )
    {
        NMSprintf
        (
            buff,
            MSG( "%2d%02d%02d%02d", 295),
            minutes,
            GbIDOSCountryInfo.timeSep[ 0 ],
            seconds,
            GbIDOSCountryInfo.decimalSep[ 0 ],
            tenths
        );
    }
    else
    {
        NMSprintf
        (
            buff,
            MSG( "%2d%02d%02d", 296),
            seconds,
            GbIDOSCountryInfo.decimalSep[ 0 ],
            tenths
        );
    }
}

void
HandleScrollablePortal( PCB *portal )
{
    int escapeFlag = 0;
    LONG    keyType;
    BYTE    ch;
    int     updatedDisplay = TRUE;

    LONG    virtualHeight;
    LONG    portalHeight;
    LONG    bottomLine;
    LONG    curPos;
    LONG    vline;

    /*
    ** The values for portal->portalHeight and portal->virtualHeight are the
    ** only two that we deal with that are 1-based. All the rest are 0-base
    */
}

```

```

h
** so we will use local copies of these variables adjusted to fit in wit
** the rest in the PCB structure.
*/
virtualHeight = portal->virtualHeight - 1;
portalHeight = portal->portalHeight - 1;

/* Other initializations.
*/
bottomline = virtualHeight - portalHeight;

while(!escapeFlag)
{
    if ( updatedisplay == TRUE )
    {
        /* The last iteration of the loop made a change, so redr
        ** portal.
        */
        if ( portal->verticalScroll == SCROLL_ON )
        {
            /* Adjust the vertical thumb on the right border
            */
            if ( portal->cursorline == 0 )
            {
                if ( virtualHeight <= portalHeight )
                {
                    curPos = 100;
                }
                else
                {
                    curPos = 0;
                }
            }
            else
            {
                if ( portal->cursorline >= bottomline )
                {
                    curPos = 100;
                }
                else
                {
                    vline = bottomline;
                    if ( vline == 0 )
                    {
                        vline = 1;
                    }
                }
            }
            curPos = ( portal->cursorline *
100 ) / vline;

        }
    }
    /* Erase the vertical thumb at its last position
    */
    portal->showScrollBars &= ~VERTICAL_SCROLL_MASK;
    /* Display the vertical thumb at its new position
    */
    n.
}

```

OOL_SHIFT;

)

```

NMUpdatePortal( portal );
updatedisplay = FALSE;

```

```

)
NMGetKey( &keyType, &ch, NUTHandle );
switch ( keyType )
{
    case K_UP:

```

```

/* Scroll the portal up one line.
*/
if ( portal->virtualline > 0 )
{

```

```

/* We're not at the top, so we can scrol
*/
portal->virtualline--;
portal->cursorline = portal->virtualline
updatedisplay = TRUE;

```

break;

case K_DOWN:

```

/* Scroll the portal down one line.
*/
if ( portal->virtualline < bottomline )
{

```

```

/* We're not at the bottom, so we can sc
*/
portal->virtualline++;
portal->cursorline = portal->virtualline
updatedisplay = TRUE;

```

break;

case K_PUP:

```

/* Move the portal up one page.
*/
if ( portal->cursorline > 0 )
{

```

LONG delta;

```

/* We're not at the top, so we can move
up. However, we need

```

```
#include "dpcagent.h" /* Our header file */
```

```
/*
 * History:
 *
 * Version 0.1 - First Package delivery for demo use.
 * Version 0.2 - (03-15-96)
 *
 * Site ID filtering to catalog
 * delivery support
 *
 * Stats()
 *
 * ig into file and added DPCUpdateConfig()
 * * Version 0.3 - (03-25-96)
 *
 * upport for interactivity field in
 *
 * ee... in catalog by adding minimum
 * ore committing entry.
 * * Version 0.4 - (03-29-96)
 *
 * indexes to modem configuration and
 *
 * o 2400 to fix key reception bugs.
 *
 * to enable agent to continue getting
 * ssful, even if agent shuts down.
 * * Version 0.5 - (04-09-96)
 *
 * showing up in Cancel Download list
 *
 * tatistics halfway running (doesn't abend)
 *
 * explicit) files in catalogue
 *
 * west/response/confirm code running but
 * it yet.
 * * Version 0.6 - (04-10-96)
 *
 * nting Get MLID stats
 *
 * alStrength()
 * * Version 0.7 - (04-11-96)
 *
 * igation Editor
 *
 * ength Meter
 * * Version 0.8 - (04-12-96)
 *
 * alStrength to report 0 when not connected
 *
 * nting cost(explicit) file download
 *
 * View Database Entries option worked
 *
 * entire entry and is more usable.
 * * Version 0.9 - (04-16-96)
 *
 *
 * yet until IJ gat
 *
 * Added ARP loopba
 *
 * Fixed missing "e
 * to GatherFragmen
 *
 * Fixed NWSAppennd
 * character sets i
 *
 * Completion of Ph
 * Places prefix in
 * Added support fo
 * Allowed init str
 * Modified IP and
 * Debug screen dis
 *
 * Added connect ba
 * Fixed Tx LED so
 * Added Adapter In
 * Fixed phone numb
 * Bumped AIO write
 *
 * Broke out LAN dr
 * Added Agent Regi
 * call us when it
 * abends.
 *
 * Added -NOPD swit
 *
 * Revised send rou
 *
 * Changed serial n
 * Added AIO buffer
 * Added auto login
 *
 * Completely rewro
 * Added Modem Cont
 *
 * Added GNU condit
 */
```

```
ionals.
*
pt timeouts
*
utines into PPP.C
*
*****
#define AGENT_VERSION InxMSG("1.20 ", 173)
/*****
* Global variable used by DPCAGENT.C
*****
/
* Resource Tag variables.
*/
struct LoadDefinitionStructure *NLMHandle = 0;
struct ResourceTagStructure *allocRTag = 0;
struct ResourceTagStructure *timerTag = 0;
struct ResourceTagStructure *AESTag = 0;
struct ResourceTagStructure *asyncIOtag = 0;
/*
* NUT and screen ID variables.
*/
NUTInfo
NUT handle
/*
* NUTHandle = NULL;
*/
/* main
*/
/* for OutputToScreen
*/
#endif
WORD
/* Screen Height(25)
*/
WORD
/* Screen Width(80)
*/
LONG
ackground Portal
/*
* Process and thread variables.
*/
BYTE
inter to messages
LONG
/* Main Handler thread PID
*/
LONG
/* Packet Handler thread PID
*/
LONG
/* Modem Handler thread PID
*/
LONG
/* Access thread PID
*/
LONG
/* Turbo Internet thread PID
*/
BYTE
int
/* All threads must exit
*/
LONG
/* Tinet needs to wake up flag
*/
int
/*
* CRC table used by calccrc().
*/
static unsigned short crctab[256] = {
0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6636, 0x74bf,
0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xcabc, 0xdb55, 0xe97e, 0xf8bf,
0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,

```

```
0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xf7a7, 0xc87c, 0xd9f5,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
0xbdc3, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8df, 0xc974,
0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0xd732, 0xc6bb,
0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
0x5285, 0x430c, 0x301e, 0x2117, 0x528, 0x437b3, 0x263a,
0xdced, 0xf44, 0xfdd, 0xec56, 0x98e9, 0x8960, 0xbfb, 0xaa72,
0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xficf, 0xee46, 0xdcdd, 0xc54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xf13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cfe,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3b3d, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0xc0c0, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0x5a5c, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
);
/* MLID Statistic globals */
int
GenericDescriptionTable[22+8] =
InxMSG("Total packets sent:", 103),
InxMSG("Total packets received:", 141),
InxMSG("No ECB available count:", 148),
InxMSG("Send packet too big count:", 156),
InxMSG("Reserved:", 186),
InxMSG("Receive packet overflow count:", 214),
InxMSG("Receive packet too big count:", 215),
InxMSG("Receive packet too small count:", 216),
InxMSG("Send packet miscellaneous errors:", 217),
InxMSG("Receive packet miscellaneous errors:", 218),
InxMSG("Send packet retry count:", 223),
InxMSG("Checksum errors:", 290),
InxMSG("Hardware receive mismatch count:", 291),
InxMSG("Total send OK byte count low:", 229),
InxMSG("Total send OK byte count high:", 230),
InxMSG("Total receive OK byte count low:", 231),
InxMSG("Total receive OK byte count high:", 232),
InxMSG("Total group address send count:", 233),
InxMSG("Total group address receive count:", 251),
InxMSG("Adapter reset count:", 252),
InxMSG("Adapter operating time stamp:", 253),
InxMSG("Adapter queue depth:", 255),
InxMSG("Send OK single collision count:", 159),
InxMSG("Send OK multiple collision count:", 256),
InxMSG("Send OK but deferred", 264),
InxMSG("Send abort from late collision", 265),
InxMSG("Send abort from excess collision", 266),
InxMSG("Send abort from carrier sense", 267),
InxMSG("Send abort from excessive deferral", 268),
InxMSG("Receive abort from bad frame alignment", 269)

```

```
);

#define STATS_DATA_WIDTH 13
#define FSD_DATA_WIDTH 40
#define BORDER_WIDTH 3
#define INDENT_WIDTH 3

void ( *BackgroundFuncPtr ) ( LONG portalNumber ) = NULL;
LONG BackgroundPortal;
int GenericLineStart;
int GblDataCol;
struct DOSCountryInfoStruct GblDOSCountryInfo;

int DebugFlag = FALSE;

/* The array nDeclination contains the Declination in degrees to add or
 * subtract from true azimuth to get magnetic azimuth.
 * The value 0 means that the location is not in mainland US
 */
static float nDeclination[] =
{
    0, 0, 4, 1, 0, 0, 0, -8, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, -3, -4, -6, -8, -10, -11, 12, -13, 0, 0, 0,
    0, 0, 5, 2, -2, -5, -6, -9, -11, -12, -13, -13, 0, 0, 0,
    0, 7, 5, 2, -1, -4, -6, -8, -12, -12, -14, -15, -16, -16,
    0, 12, 9, 6, -1, -4, -6, -9, -11, -12, -14, -16, -17, -17,
    18, 15, 0, 7, 2, -3, -6, -9, -12, -13, -15, -18, -19, -19,
    0, 0, 0, 0, 0, -1, -6, -9, -13, -15, -18, -20, -21, -22
};

#ifdef LOG_ECB_ACTIVITY
int DPC_TGID;
LogHandle LogClientHandle;
EventHandle LogECBHandle;
#endif /* LOG_ECB_ACTIVITY */

/*
 * Local function prototypes.
 */
void ReturnResources(int sig);
/*
 * Exithandler(void *handle)
 *
 * Description:
 * This routine is called when the user hits Alt-F10 to exit
 * the utility. We will attempt to verify with the user that
 * they really want to exit.
 *
 * Input: handle - NUT handle
 *
 * Output: nothing
 *
 * Returns: nothing
 */
void Exithandler(void *handle)
{
    if (NWSConfirm(InMsg("Exit DSDEBUG?", 88), 0, 0, TRUE, NULL,
        handle, NULL) == TRUE)

```

```
    {
        ReturnResources(1);
    }
    exit(1);
}

static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
{
    head = head;
    tail = tail;
    handle = handle;
    return;
}

/*
 * calccrc(WORD crc,
 *         BYTE *cp,
 *         LONG len)
 *
 * Description:
 * This routine calculates a CRC value for the text passed
 * in. It uses a CRC substitution table for efficiency.
 * Its used by the Slip transmit routine.
 *
 * Input:  crc
 *         - Initial value of the C
 *        RC.
 *        cp
 *        - string to calc
 *        len
 *        - length of the string
 *
 * Output: nothing
 *
 * Returns: 16-bit CRC value
 */
WORD calccrc(WORD crc, BYTE *cp, LONG len)
{
    while(len--)
        crc = (crc >> 8) ^ crctab[(crc ^ *cp++) & 0xff];
    return(crc);
}

/*
 * SlipSend(char *pdata,
 *          LONG sz,
 *          int cas,
 *          int timeout)
 *
 * Description:
 * This routine builds a SLIP envelope around the message
 * passed in, escapes any HDLC characters in the message,
 * and sends it to the modem.
 *
 * Input:  pdata
 *        sz
 *        - Pointer to the message
 *        - length of the
 *        message passed in
 */

```



```

h
** so we will use local copies of these variables adjusted to fit in wit
** the rest in the PCB structure.
**/
virtualHeight = portal->virtualHeight - 1;
portalHeight = portal->portalHeight - 1;
/* Other initializations.
**/
bottomLine = virtualHeight - portalHeight;
while(!escapeFlag)
{
    if ( updatedDisplay == TRUE )
    {
        /*
        ** The last iteration of the loop made a change, so redr
        ** portal.
        **/
        if ( portal->verticalScroll == SCROLL_ON )
        {
            /*
            ** Adjust the vertical thumb on the right border
            **/
            if ( portal->cursorLine == 0 )
            {
                if ( virtualHeight <= portalHeight )
                    curPos = 100;
                else
                    curPos = 0;
            }
            else
            {
                if ( portal->cursorLine >= bottomLine )
                    curPos = 100;
                else
                {
                    vline = bottomLine;
                    if ( vline == 0 )
                        vline = 1;
                }
            }
            curPos = ( portal->cursorLine *
100 ) / vline;

            /* avoid divide by 0 */
            curPos = ( portal->cursorLine *
100 ) / vline;

            /* Erase the vertical thumb at its last position
            **/
            portal->showScrollBars &= ~VERTICAL_SCROLL_MASK;
            /*
            ** Display the vertical thumb at its new position
            **/
            portal->showScrollBars |= VERTICAL_SCROLL_MASK;
        }
    }
}

```

n.

```

/*
**
**/
portal->showScrollBars |= curPos << VERTICAL_SCR
OLL_SHIFT;
}
NWSUpdatePortal( portal );
updatedDisplay = FALSE;
}
NWSGetKey( &keyType, &ch, NUTHandle );
switch ( keyType )
{
    case K_UP:
        /*
        ** Scroll the portal up one line.
        **/
        if ( portal->virtualLine > 0 )
        {
            /*
            ** We're not at the top, so we can scrol
            **/
            portal->virtualLine--;
            portal->cursorLine = portal->virtualLine
            updatedDisplay = TRUE;
        }
        break;
    case K_DOWN:
        /*
        ** Scroll the portal down one line.
        **/
        if ( portal->virtualLine < bottomLine )
        {
            /*
            ** We're not at the bottom, so we can sc
            **/
            portal->virtualLine++;
            portal->cursorLine = portal->virtualLine
            updatedDisplay = TRUE;
        }
        break;
    case K_PUP:
        /*
        ** Move the portal up one page.
        **/
        if ( portal->cursorLine > 0 )
        {
            LONG delta;
            /*
            ** We're not at the top, so we can move
            **/
            delta = portal->virtualLine - portal->cursorLine;
            portal->virtualLine = portal->virtualLine - delta;
            portal->cursorLine = portal->cursorLine - delta;
            updatedDisplay = TRUE;
        }
        break;
}

```

up.

```

** to figure out how much we can move up
*/
if ( portal->cursorLine > portalHeight )
    delta = portalHeight;
else
    delta = portal->cursorLine;

portal->cursorLine -= delta;
if ( portal->cursorLine < portal->virtua
    portal->virtualLine = portal->cu

    updatedisplay = TRUE;

        )

        case K_PDOWN:
            /* Move the portal down one page.
            */
            if ( portal->cursorLine < virtualHeight )
            {
                LONG delta;
                LONG newCurrentLine;

                /* We're not at the bottom, so we can mo
                ** we need to figure out how much we can
                */
                delta = virtualHeight - portal->cursorLi

                if ( delta > portalHeight )
                    delta = portalHeight;

                newCurrentLine = portal->cursorLine + de

                delta = virtualHeight - portalHeight;
                if ( newCurrentLine > delta )
                    portal->virtualLine = delta;
                else
                    portal->virtualLine = newCurrent

                portal->cursorLine = portal->virtualLine

                updatedisplay = TRUE;
            }
            break;

        case K_SUP:
            /*
            ** <Ctrl-PgUp> takes us to the top of the portal
            */
            portal->virtualLine = 0;
            portal->cursorLine = 0;
            updatedisplay = TRUE;
        }

        void UpdateStatsInformation(LONG portal)
        {
            PCB
            struct DriverStatsStructure *stats;

```

```

break;

        case K_SDOWN:
            /*
            ** <Ctrl-PgDn> takes us to the bottom of the por
            */
            portal->cursorLine = virtualHeight;
            portal->virtualLine = portal->cursorLine - porta

            updatedisplay = TRUE;
            break;

        case K_ESCAPE:
            escaperflag = 1;
            break;

        )

        DPCGetSignalStrength(void)
        {
            struct DriverStatsStructure *stats;
            CustVars *customPtr;
            LONG signal;
            int beamSize, beamPercent;

            if (DPCGetMLIDStats(&stats))
                return(0);

            customPtr = (CustVars *)(&stats->CustomVariableCount);
            signal = customPtr->CustomVariable[0];

            if (signal >= 200)
                signal = (2 * (signal - 200)) + 60;
            else
                signal = 0;

            beamSize = (int)((signal - 60L)/2L);
            beamSize *= 5;
            beamSize /= 4;

            if (beamSize < 0)
            {
                beamSize = 0;
            }
            else if (beamSize >= MAX_BEAM)
            {
                beamSize = MAX_BEAM;
            }

            beamPercent = (100*beamSize) / MAX_BEAM;
            return(beamPercent);
        }

        void UpdateStatsInformation(LONG portal)
        {
            PCB
            struct DriverStatsStructure *stats;

```

```

int line, count;
LONG
int numGenerics;
BYTE
mask;
LONG
seconds;
LONG
tenths;
CustVars
*customPtr;
BYTE
NWSGetPCB( &portalPtr, portal, NUTHandle );

if ( DPCGetMLIDStats(&stats))
(
    AddKey( NUTHandle->screenID, ENTER_KEY, 0, 0, 0 );
    return;
)

/* do generic stuff */
line = GenericLineStart;
statsPtr = &( stats->NotSupportedMask );
numGenerics = stats->GenericVariableCount;
mask = *statsPtr++;

for ( count = 0; count < numGenerics; count++ )
(
    if ( mask & ( 0x80000000 >> ( count & 0x1f ) ) )
    (
        /* not supported */
        NWSprintf( string, MSG("%13.13s", 301), MSG("Not support
ed", 298) );

        NWSShowPortalline
        (
            line++,
            GblDataCol,
            string,
            STATS_DATA_WIDTH,
            portalPtr
        );
        statsPtr++;
    )
    else
    (
        if ( count == 20 )
        {
            BYTE tmpString[ 80 ];

            /* This is the operating time stamp, which needs
            ** output format.
            */
            ConvertTicksToSeconds( *statsPtr++, &seconds, &t
            enths );
            FormatElapsedTime( seconds, tenths, tmpString );
            NWSprintf( string, MSG("%13.13s", 302), tmpStrin
            g );
        }
        else
        (
            CreateStringWithCommas
            (
                *statsPtr++,
                string,
                MSG("%13lu", 303)
            );
        )
        NWSShowPortalline
        (
            line++,
            GblDataCol,
            string,
            STATS_DATA_WIDTH,
            portalPtr
        );
    )
)

/* do custom stuff */
line += 2;
customPtr = (CustVars *)(&stats->CustomVariableCount);
ptr = (BYTE *) (customPtr->CustomVariable[customPtr->CustomVariableCount]
);
ptr += sizeof(WORD);

for ( count = 0; count < customPtr->CustomVariableCount; count++ )
(
    CreateStringWithCommas
    (
        customPtr->CustomVariable[ count ],
        string,
        MSG("%13lu", 299)
    );
    NWSShowPortalline( line++, GblDataCol, string, STATS_DATA_WIDTH,
portalPtr );
}

NWSUpdatePortal( portalPtr );

void SignalMeter(void) {
    int exitNow = FALSE;
    LONG type;
    BYTE value;
    int signal = 0;
    int oldSignal = 0;
    int avgSqf = 0;
    int beamSize;
    int beamPercent;
    int bBlock = FALSE;
    int rxFreq;
    struct DriverStatsStructure *stats;
    CustVars
    *customPtr;
    char freqStr[80];
    char aveStr[80];
    char signalStr[80];
    int sound_gap = 0;

    while(!exitNow) {
        if (DPCGetMLIDStats(&stats)) {
            type = signal = 0;
            rxFreq = 0;
        }
        else {
            customPtr = (CustVars *)(&stats->CustomVariableCount);
            type = signal = customPtr->CustomVariable[0];
        }
    }
}

```

```

    rxFreq = customPtr->CustomVariable[1];
}

NWSprintf(freqStr, MSG("      Frequency : %d", 345), rxFreq / 10);

if (signal >= 200)
    signal = (2 * (signal - 200)) + 60;
else
    signal = 0;

if (avgSqf == 0L)
    avgSqf = 100L * signal;
avgSqf = ((avgSqf * 19L) + (100L * (unsigned long) signal)) / 20L;

beamSize = (int)((avgSqf/100L - 60L)/2L);
beamSize *= 5;
beamSize /= 4;

if (beamSize < 0)
    beamSize = 0;
else if (beamSize >= MAX_BEAM)
    beamSize = MAX_BEAM - 1;

beamPercent = (100*beamSize) / MAX_BEAM;

if (oldSignal != signal) {
    if (signal < MIN_SOF_VAL)
        block = FALSE;
    else
        block = TRUE;
    oldSignal = signal;
}

NWSprintf(aveStr, MSG("      Average SOF : %d", 346),
    signal ? avgSqf / 100L : 0);

NWSprintf(signalStr, MSG("Signal Quality : %d (%d%%)",
    signal, 347),
    signal,
    beamPercent,
    block ? MSG("(Signal Locked)", 348) : MSG("(Signal Not Locked)", 349));

DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 341),
    beamSize,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);

if (--sound_gap <= 0) {
    /* calculate frequency based on raw signal strength */
    signal = 200 + type;
    /* adjust for 8253-5 timer chip output */
    signal = 1193180 / signal;

    /* turn on sound */
    outp(67, 182);
    outp(66, signal & 256);
    outp(66, signal / 256);
    outp(97, inp(97) | 0x03);

    delay(300);

    /* turn off sound */
    outp(97, inp(97) & ~0x03);
}

    sound_gap = (110 - beamPercent) / 17;
}

delay(150);

if (NWSKeyStatus(NUTHandle)) {
    NWSGetKey(&type, &value, NUTHandle);
    if ((type == K_ESCAPE) || (type == K_AF10))
        {
            exitNow = TRUE;
        }
}

/* Destroy the throttle portal */
DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 351),
    MAX_BEAM,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);
}

void GetRegionName(char *regionName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 635), MSG("r", 636));
    len = strlen(MSG("RegionName=", 637));
    *regionName = 0;
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(szTmp, MSG("RegionName=", 638), len)) ==
                (
                    fclose(fp);
                    src = &szTmp[len];
                    dest = regionName;
                    while(*src != 0 && *src != ' ' && *src != '\r' &
                        {
                            *dest = *src;
                            src++;
                            dest++;
                        }
                    *dest = 0;
                    return;
                )
            )
                fclose(fp);
        }
    }

    void GetCountry(char *countryName)
    {
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;
    }
}

```

```

fp = fopen(MSG("country.ini", 639), MSG("r", 640));
len = strlen(MSG("Name=", 641));
*countryName = 0;
if (fp != NULL)
{
    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        if ( (strnicmp(szTmp, MSG("Name=", 642), len)) == 0)
        {
            fclose(fp);
            src = &szTmp[len];
            dest = countryName;
            while(*src != 0 && *src != ' ' && *src != '\r' &
                {
                    *dest = *src;
                    src++;
                    dest++;
                }
            *dest = 0;
            return;
        }
        fclose(fp);
    }
}

void GetDefaultService(char *serviceName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 626), MSG("r", 627));
    len = strlen(MSG("Service=", 628));
    *serviceName = 0;
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(szTmp, MSG("Service=", 185), len)) == 0)
            {
                fclose(fp);
                src = &szTmp[len];
                dest = serviceName;
                while(*src != 0 && *src != ' ' && *src != '\r' &
                    {
                        *dest = *src;
                        src++;
                        dest++;
                    }
                *dest = 0;
                return;
            }
            fclose(fp);
        }
    }

    typedef struct
    {
        char name[20];
        LONG longitude;
        LONG eastFlag;
        LONG frequency;
        LONG horzFlag;
        LONG cityLongDegrees;
        LONG cityLongMinutes;
        LONG cityLatDegrees;
        LONG cityLatMinutes;
        LONG cityEastFlag;
        LONG cityNorthFlag;
    } SatelliteInfo;

    typedef struct
    {
        float elevation;
        float trueAzimuth;
        float magAzimuth;
        float polarization;
    } DishInfo;

    void ParseSatelliteInfo(char *satName, int nameContainsInfo, SatelliteInfo *sat)
    {
        FILE *fp;
        char szTmp[128];
        char *fpPtr;
        char *name, *cptr;
        char *ascPtr;
        char ascBuf[10];

        if (nameContainsInfo == FALSE)
        {
            fp = fopen(MSG("country.ini", 644), MSG("r", 629));
            if (fp != NULL)
            {
                while ((fpPtr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != N
                    ULL)
                {
                    if ( (strnicmp(szTmp, satName, strlen(satName)))
                        == 0)
                    {
                        break;
                    }
                    if (fp)
                        fclose(fp);
                    if (!fpPtr)
                        return;
                    cptr = szTmp;
                }
                else
                {
                    cptr = satName;
                    name = sat->name;
                    while(*cptr != '\n')
                        *name++ = *cptr++;
                    *name = 0;
                    cptr++;
                    while(*cptr == ' ')
                        cptr++;
                    ascPtr = ascBuf;
                    while(*cptr != ',')
                        *ascPtr++ = *cptr++;
                }
            }
        }
    }

```



```

    NewCalculationFlag = 1;
    rcode = K_ESCAPE;
}

ChangesSatExit:
    NWSDestroyList (NUTHandle);
    NWSPopList (NUTHandle);
    NWSSetListSortFunction (NUTHandle, oldSortFunction);
    return rcode;
}

LONG
{
    ComputeHotSpot (FIELD *fp, int key, int *changed, NUTInfo *handle)

    fp = fp;
    key = key;
    changed = changed;
    handle = handle;

    NewCalculationFlag = 1;
    return K_ESCAPE;
}

int
{
    LongitudeHemisphereHandler (int option, void *parameter)

    parameter = parameter;
    return option;
}

int
{
    PolarizationHandler (int option, void *parameter)

    parameter = parameter;
    return option;
}

int
{
    GroundLatHemHandler (int option, void *parameter)

    parameter = parameter;
    return option;
}

int
{
    GroundLongHenHandler (int option, void *parameter)

    parameter = parameter;
    return option;
}

float SGN (float num)
{
    if (num < 0)
        return -1;
    return 1;
}

void Calculatedish (SatelliteInfo *sat, DishInfo *dish)
{
    float LW, LN, ZR, YR, XR, DIST, EL, PO, AZ, TRAZ, S, A;
    float SD, RD, RM, LD, LM;
    static const float M_PI = 3.14159;
    static const float RAD = 6378.388;
    static const float ALT = 42164.24;
    int count = 4;

    SD = p->dSatLong;
    RD = p->dRemLongDeg;
    RM = p->dRemLongMin;

```

```

//
//
LD = p->dRemLatDeg;
LM = p->dRemLatMin;

SD = (float) sat->longitude;
RD = (float) sat->cityLongDegrees;
RM = (float) sat->cityLongMinutes;
LD = (float) sat->cityLatDegrees;
LM = (float) sat->cityLatMinutes;

SD = fabs (SD);
if (p->cSatLong == 'E')
    if (sat->eastFlag)
        SD = -SD;

RD = fabs (RD);
RM = fabs (RM);
if (p->cRemLong == 'E')
    if (sat->cityEastFlag)
    {
        RD = -RD;
        RM = -RM;
    }

LD = fabs (LD);
LM = fabs (LM);

if (p->cRemHem == 'S')
    if (sat->cityNorthFlag == 0)
    {
        LD = -LD;
        LM = -LM;
    }

LW = (RD + RM / 60) - SD * M_PI / 180;

if (LW == 0)
    LW = .00001;
LN = (LD + LM / 60) * M_PI / 180;

if (LN == 0)
    LN = .00001;

ZR = RAD * sin (LN);
YR = RAD * cos (LN) * cos (LW);
XR = RAD * cos (LN) * sin (LW);
DIST = sqrt (XR * XR + (ALT - YR) * (ALT - YR) + ZR * ZR);

EL = (ALT * YR - RAD * RAD) / (RAD * DIST);
EL = atan (EL / sqrt (1 - EL * EL)) * 180 / M_PI;

EL = (10 * EL + .5 * SGN (EL)) / 10;

A = 0;

if (LN * LW > 0)
    A = 2 * M_PI;
if (LN > 0)
    A = M_PI;

AZ = (A - atan (tan (LW) / sin (LN))) * 180 / M_PI;
AZ = floor (10 * AZ + .5 * SGN (AZ)) / 10;
TRAZ = AZ;

/* Executed for US Mainland only */
/* Declination correction to TRUE Azimuth */
if (((LD > 23) && (LD < 50)) && ((RD > 70) && (RD < 125)))

```

```

(
    LD = floor(LD/4) - 6;
    RD = ceil(RD/4) - 18;
    if(!nDeclination((int)(LD*14+RD)))
        count--;
    if(!nDeclination((int)(LD*14+RD-1)))
        count--;
    if(!nDeclination((int)((LD+1)*14+RD-1)))
        count--;
    if(!nDeclination((int)((LD+1)*14+RD)))
        count--;
    if(count)
    {
        AZ = AZ + nDeclination((int)(LD*14+RD))
            + nDeclination((int)(LD*14+RD-1))
            + nDeclination((int)((LD+1)*14+RD))
            + nDeclination((int)((LD+1)*14+RD-1))/count;
    }

    if (AZ >= 360) AZ = AZ - 360;

    if (AZ < 0 ) AZ = AZ + 360;

    S = tan(LN) / sin(LW);

    PO = atan(S);
    /* printf("S=%f LN=%f LW=%f PO=%f\n",S,LN,LW,PO); */

    PO = fabs(PO);
    PO = M_PI / 2.0 - PO;
    PO = PO * SGN(S) * 180 / M_PI;
    PO = floor(10 * PO + .5 * SGN(PO)) / 10;

    //
    //
    //
    //
    p->dRemElev = EL;
    p->dRemMagAz = AZ;
    p->dRemTrueAz = TRAZ;
    p->dRemPolar = -PO;
    dish->elevation = EL;
    dish->magAzimuth = AZ;
    dish->trueAzimuth = TRAZ;
    dish->polarization = -PO;
}

void ComputeCity(char *region, char *city, LONG latLong)
(
    FIELD *fp;
    char country[20], countryStr[30];
    char regionStr[30], cityStr[30];
    char service[30], serviceStr[30];
    char satelliteStr[50];
    char elevationStr[50], trueAzimuthStr[50];
    char magAzimuthStr[50], polarizationStr[50];
    SatelliteInfo sat;
    DishInfo dish;
    int i;
    int start;
    MFCNTROL *mfctl0, *mfctl1, *mfctl2, *mfctl3;

    calculateAgain;
    NWSInitForm(NUTHandle);

    if (NewCalculationFlag == 0)
    {

```

```

        sat.cityLatDegrees = (latLong >> 24) & 0xff;
        sat.cityLatMinutes = (latLong >> 16) & 0xff;
        sat.cityLongDegrees = (latLong >> 8) & 0xff;
        sat.cityLongMinutes = (latLong >> 0) & 0xff;
        GetDefaultSatellite(&sat);
        sat.cityEastFlag = sat.eastFlag;
        sat.cityNorthFlag = 1;
    }
    NewCalculationFlag = 0;

    i = 0;
    GetCountry(country);
    NWSprintf(countryStr, MSG("Country : %s", 488), country);
    NWSAppendCommentField(i, 2, countryStr, NUTHandle);

    NWSprintf(regionStr, MSG("Region : %s", 712), region);
    NWSAppendCommentField(i, 36, regionStr, NUTHandle);

    i++;
    NWSprintf(cityStr, MSG("City : %s", 713), city);
    NWSAppendCommentField(i, 2, cityStr, NUTHandle);

    GetDefaultService(service);
    NWSprintf(serviceStr, MSG("Service : %s", 714), service);
    NWSAppendCommentField(i, 36, serviceStr, NUTHandle);

    i++;
    NWSprintf(satelliteStr, MSG("Satellite : %s", 649), sat.name);
    start = (78 - strlen(satelliteStr)) / 2;
    fp = NWSAppendHotSpotField(i, start, NORMAL_FIELD, satelliteStr,
        ChangesSatellite, NUTHandle);
    fp->customData = &sat;

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Longitude : ", 715), NUTHa
        ndle);
    NWSAppendIntegerField(i, 27, NORMAL_FIELD, (int *)&sat.longitude, 1, 180
        , F_NO_HELP, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Hemisphere : ", 716), NUTHandle);
    mfctl0 = NWSInitMenuField(InxMSG("Satellite Longitude Hemisphere", 717),
        10, 40, LongitudeHemisphereHandler, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("West", 718), 0, NUTHandle);
    NWSAppendToMenuField(mfctl0, InxMSG("East", 719), 1, NUTHandle);
    NWSAppendMenuField(i, 47, NORMAL_FIELD, (int *)&sat.eastFlag, mfctl0, NU
        LL, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Polarization : ", 489), NUTHa
        ndle);
    mfctl1 = NWSInitMenuField(InxMSG("Satellite Polarization", 490), 10, 40,
        PolarizationHandler, NUTHandle);
    NWSAppendToMenuField(mfctl1, InxMSG("Vert", 531), 0, NUTHandle);
    NWSAppendToMenuField(mfctl1, InxMSG("Horz", 539), 1, NUTHandle);
    NWSAppendMenuField(i, 27, NORMAL_FIELD, (int *)&sat.horzFlag, mfctl1, NU
        LL, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Frequency : ", 541), NUTHandle);
    NWSAppendIntegerField(i, 47, NORMAL_FIELD, (int *)&sat.frequency, 1, 100
        00, F_NO_HELP, NUTHandle);

    i+=2;
    NWSAppendCommentField(i, 2, MSG("Ground Longitude degrees : ", 543), NUT
        Handle);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLongDegrees,
        1, 180, F_NO_HELP, NUTHandle);
}

```

```

NWSAppendCommentField(i, 40, MSG("minutes : ", 545), NUTHandle);
NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLongMinutes,
    1, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 586), NUTHandle);
mfc12 = NWSInitMenuField(InxMSG("Ground Longitude Hemisphere", 687), 10
    , 40, GroundLongHandler, NUTHandle);
NWSAppendToMenuField(mfc12, InxMSG("West", 688), 0, NUTHandle);
NWSAppendToMenuField(mfc12, InxMSG("East", 689), 1, NUTHandle);
NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityEastFlag, mfc12
    , NULL, NUTHandle);

i++;
NWSAppendCommentField(i, 2, MSG("Ground Latitude degrees : ", 690), NUT
    Handle);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLatDegrees, 1
    , 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 40, MSG("minutes : ", 691), NUTHandle);
NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLatMinutes, 1
    , 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 692), NUTHandle);
mfc13 = NWSInitMenuField(InxMSG("Ground Latitude Hemisphere", 693), 10,
    40, GroundLatHemHandler, NUTHandle);
NWSAppendToMenuField(mfc13, InxMSG("South", 694), 0, NUTHandle);
NWSAppendToMenuField(mfc13, InxMSG("North", 695), 1, NUTHandle);
NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityNorthFlag, mfc1
    3, NULL, NUTHandle);

i++;
NWSAppendHotSpotField(i, 35, NORMAL_FIELD, MSG("COMPUTE NOW", 696),
    ComputeHotSpot, NUTHandle);

CalculatedDish(&sat, &dish);

if (sat.horzFlag == 0)
{
    dish.polarization += 90.0;
    if (dish.polarization > 90.0)
        dish.polarization -= 90.0;
    if (dish.polarization < -90.0)
        dish.polarization += 90.0;
}

i+=2;

NWSprintf(elevationStr, MSG(" Elevation : %lf", 697), dish.elev
    ation);
NWSprintf(trueAzimuthStr, MSG(" True Azimuth : %lf", 698), dish.true
    Azimuth);
// if ((strcmp(country, MSG("USA", 243))) != 0)
//     strcpy(magAzimuthStr, MSG("Magnetic Azimuth : N/A", 699));
// else
//     NWSprintf(magAzimuthStr, MSG("Magnetic Azimuth : %lf", 568), d
    ish.magAzimuth);
NWSprintf(polarizationStr, MSG(" Polarization : %lf", 700), dish.pola
    rization);

NWSAppendCommentField(i, 2, elevationStr, NUTHandle);
i++;
NWSAppendCommentField(i, 2, trueAzimuthStr, NUTHandle);
i++;
NWSAppendCommentField(i, 2, magAzimuthStr, NUTHandle);
i++;

```

```

NWSAppendCommentField(i, 2, polarizationStr, NUTHandle);
i++;

NWSEditPortalForm(InxMSG("Antenna Pointing Calculations", 701),
    12, 40,
    /* center line, column */
    i, 78,
    /* form height, width */
    F_NOVERIFY, F_NO_HELP,
    /* Control flags, help m
    essage */
    NULL,
    NUTHandle);

le /*

NWSDestroyForm(NUTHandle);
if (NewCalculationFlag != 0)
    goto calculateAgain;
}

void ComputeGetCity(char *region)
{
    DIR *dirCountry, *dirCity;
    char *name;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;
    FILE *fp;
    char szTmp[128];
    char byteStr[8];
    int latDeg, latMin, longDeg, longMin;
    int len, start = 0;
    LONG info;

    NWSStartWait( 0, 0, NUTHandle ); /* DWH change 970131 */

    getCitiesLoop:
    rows = cols = 0;
    listPtr = NULL;
    NWSInitList(NUTHandle, NULL);

    dirCountry = opendir(MSG("*.cty", 702));
    if (dirCountry == NULL)
        goto errorNoDir;

    while( (dirCity = readdir(dirCountry)) != NULL )
    {
        name = dirCity->d_name;
        fp = fopen(name, MSG("r", 703));
        if (fp == NULL)
        {
            closedir(dirCountry);
            goto errorNoDir;
        }

        len = strlen(region);
        if (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(region, szTmp, len)) == 0 )
                break;
        }
        fclose(fp);
    }
}

```

```

if (dirCity == NULL)
{
    closedir(dirCountry);
    goto errorNoDir;
}

while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
{
    int len = strlen(szTmp) - 2;
    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    byteStr[3] = 0;
    longMin = atoi(&byteStr[1]);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    byteStr[0] = szTmp[len--];
    longDeg = atoi(&byteStr);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    latMin = atoi(&byteStr[1]);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    latDeg = atoi(&byteStr[1]);

    if (longMin > 59)
    {
        longDeg++;
        longMin = 0;
    }

    if (latMin > 59)
    {
        latDeg++;
        latMin = 0;
    }

    info = ((latDeg & 0xff) << 24) |
            ((latMin & 0xff) << 16) |
            ((longDeg & 0xff) << 8) |
            (longMin & 0xff);

    while(szTmp[len] == ' ')
        len--;
    if (len > 0)
    {
        szTmp[len+1] = 0;

        while(len)
        {
            if (szTmp[len] == ' ')
                start = len + 1;
            len--;
        }
        if (start > 0)
        {
            AppendToList(&szTmp[start], info, &rows, &cols);
        }
    }
    fclose(fp);
    if (rows == 0)
    {

```

```

        closedir(dirCountry);
        goto errorNoDir;
    }

    NWSWait(NUTHandle); /* DWH change 970131 */
    ccode = NWSList(
        InxMSG("Choose A City", 704),
        12, 40,
        (rows < 16) ? rows : 16,
        (cols < 18) ? 18 : cols,
        M_ESCAPE | M_SELECT,
        &listPtr,
        NUTHandle, NULL,
        NULL, NULL);

    if (ccode == M_SELECT)
    {
        info = (LONG)listPtr->otherInfo;
        strcpy(szTmp, listPtr->text);
        NWSDestroyList(NUTHandle);
        closedir(dirCountry);
        ComputeCity(region, szTmp, info);
        goto getCitiesLoop;
    }

    NWSDestroyList(NUTHandle);
    closedir(dirCountry);
    return;

errorNoDir:
    NWSWait(NUTHandle); /* DWH change 970131 */
    NWSDestroyList(NUTHandle);
    NWSAlert(12, 40, NUTHandle, InxMSG("Unable to access Cities file", 705));
    return;
}

void ComputeGetRegion(char *country)
{
    DIR *dirCountry, *dirCity;
    char *name, *end;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;
    char path[64];
    FILE *fp;
    char szTmp[128];
    LONG header;

    NWSWait(0, 0, NUTHandle); /* DWH change 970131 */
    getRegionsLoop:
        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);
        SetCurrentNameSpace(DOSNameSpace);
        NWSprintf(path, MSG("SYS:DIRECPC\\DB\\%s.cou", 706), country);
        if (chdir(path))
            goto errorNoDir;

        dirCountry = opendir(MSG("**.cty", 707));
        if (dirCountry == NULL)
            goto errorNoDir;

```

```

while( (dirCity = readdir(dirCountry)) != NULL )
{
    name = dirCity->d_name;
    fp = fopen(name, MSG("r", 708));
    if (fp == NULL)
    {
        closedir(dirCountry);
        goto errorNoDir;
    }
    if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        end = &szTmp[strlen(szTmp) - 2];
        while( (*end == ' ') && (end != szTmp) )
            end--;
        end++;
        *end = 0;
        AppendToList(szTmp, 0, &rows, &cols);
        fclose(fp);
    }
    if (rows == 0)
    {
        closedir(dirCountry);
        goto errorNoDir;
    }
    GetRegionName(szTmp);
    if ( (strcmpi(szTmp, MSG("Province", 709))) == 0)
        header = InxMSG("Choose A Province", 710);
    else
        header = InxMSG("Choose A State", 711);

    if (rows == 1)
    {
        NWSendWait( NUTHandle );
        NWSDestroyList(NUTHandle);
        ComputedGetCity(szTmp);
        closedir(dirCountry);
        return;
    }
    else
    {
        NWSendWait( NUTHandle );
        ccode = NWSList(
            header,
            12, 40,
            (rows < 16) ? rows : 16,
            (cols < 18) ? 18 : cols,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        /* Height */
        /* Width */

        if (ccode == M_SELECT)
        {
            strcpy(szTmp, listPtr->text);
            NWSDestroyList(NUTHandle);
            closedir(dirCountry);
            ComputedGetCity(szTmp);
            goto getRegionsLoop;
        }
    }
}

```

```

NWSDestroyList(NUTHandle);
closedir(dirCountry);
return;

errorNoDir:
NWSendWait( NUTHandle );
NWSDestroyList(NUTHandle);
NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country files in Co
untry directory %s.cou", 569), country);
return;
}

void ComputeCoordinates(void)
{
    DIR *dirDB, *dirCountry;
    char *name, *dot;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;

    getCountriesLoop:
    rows = cols = 0;
    listPtr = NULL;
    NWSInitList(NUTHandle, NULL);
    SetCurrentNameSpace(DOSNameSpace);
    if (chdir(MSG("SYS:DIREPCP\DB", 570)))
        goto errorNoDir;

    dirDB = opendir(MSG("*.cou", 571));
    if (dirDB == NULL)
        goto errorNoDir;

    while( (dirCountry = readdir(dirDB)) != NULL )
    {
        name = dirCountry->d_name;
        if ((dot = strchr(name, '.')) != NULL)
            *dot = 0;
        AppendToList(name, 0, &rows, &cols);
    }

    if (rows == 0)
    {
        closedir(dirDB);
        goto errorNoDir;
    }

    ccode = NWSList(
        InxMSG("Choose A Country", 572),
        12, 40,
        rows,
        18,
        M_ESCAPE | M_SELECT,
        &listPtr,
        NUTHandle, NULL,
        NULL, NULL);

    if (ccode == M_SELECT)
    {
        if (NWSPushList(NUTHandle) != 0)
        {
            name = listPtr->text;

```

```

        ComputeGetRegion(name);
        NWSPopList(NUTHandle);
    }
    NWSDestroyList(NUTHandle);
    closedir(dirDB);
    goto getCountriesLoop;
}

NWSDestroyList(NUTHandle);

closedir(dirDB);
return;

errorNoDir:
    NWSDestroyList(NUTHandle);
    NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country directories
    in SYS:DIREPC\IDB", 573));
    return;
}

void DPCPointing(void)
{
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;

    NWSInitList(NUTHandle, NULL);

    NWSAppendToList(MSG("Antenna Pointing Calculations", 574), (void *)1, NUTHandle);
    NWSAppendToList(MSG("Signal Strength Meter", 575), (void *)2, NUTHandle);

    while (ccode != M_ESCAPE)
    {
        ccode = NWSList(
            InxMSG("Dish Pointing", 576),
            12, 40,
            2,
            30,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        /* Height
        /* Width

        if (ccode == M_SELECT)
        {
            if (NWSPushList(NUTHandle) != 0)
            {
                switch ((int)listPtr->otherInfo)
                {
                    case 1:
                        ComputeCoordinates();
                        break;

                    case 2:
                        SignalMeter();
                        break;

                    default:
                        break;
                }
            }
            NWSPopList(NUTHandle);
        }
    }
}

```

```

    NWSDestroyList(NUTHandle);
}

void UpdateAdapterInformation(LONG portal)
{
    PCB
    int *portalPtr;
    MUXdcau_t *dptr;
    BYTE string[80];
    char serialNumber[10];

    NWSGetPCB(&portalPtr, portal, NUTHandle);

    /* do generic stuff */
    line = 0;

    DIOGetSN(serialNumber);
    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        serialNumber,
        CStrLen(serialNumber),
        portalPtr
    );

    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        SiteID,
        CStrLen(SiteID),
        portalPtr
    );

    NWSprintf(string, MSG("%s", 503), CASDBdcau.entries == 0 ? MSG("FALSE",
    474) : "TRUE ");
    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        string,
        CStrLen(string),
        portalPtr
    );

    count = CASDBdcau.entries;
    NWSprintf(string, MSG("%d", 495), count);
    NWSShowPortalLine
    (
        line++,
        GblDataCol,
        string,
        CStrLen(string),
        portalPtr
    );

    dptr = (MUXdcau_t *)CASDBdcau.p_buffer;
    while (count)
    {
        line++;
        if (line > (portalPtr->virtualHeight - 1))
            break;
    }
}

```

```

    for (col = 8; col <= 64; col += 16, count--)
    {
        if (!count)
            break;

        NWSprintf( string, MSG("%2.2X%2.2X%2.2X%2.2X", 504),
            dptr->groupid.i[2], dptr->groupid.i[1],
            dptr->groupid.i[0], dptr->version);

        NWSShowPortalLine(
            line,
            col,
            string,
            CStrLen(string),
            portalPtr
        );

        dptr++;
    }

    NWSUpdatePortal( portalPtr );
}

void DisplayAdapterInfo(void)
{
    int     line, len;
    BYTE    oldPortal;
    BYTE    *portalPtr;
    BYTE    string[80];

    line = 5 + 3; /* Site ID, S/N, Key Status, Number of Communities,
                   Current Communities,
                   extra community lines */
    line += (CASDBdcau.entries / 4);

    oldPortal = NUTHandle->currentPortal;
    NWSDeselectPortal( NUTHandle );
    BackgroundPortal = NWSCreatePortal
    (
        1 /* gblUPFTopLine */,
        1 /* gblUPFLeftCol */,
        ((line + 4) > (ScreenHeight - 3)) ? ScreenHeight - 3 : line + 4,
        /* gblUPFHeight + gblUPFHeight */
        ScreenWidth - 2 /* gblUPFWidth */,
        line,
        ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
        TRUE,
        MSG("DPC Adapter Information", 502),
        VNORMAL,
        SINGLE,
        VINTENSE,
        CURSOR_OFF,
        VIRTUAL,
        NUTHandle
    );

    if ( BackgroundPortal > MAXPORTALS )
    {
        NWSSelectPortal( oldPortal, NUTHandle );
        return;
    }

    NWSPGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
    portalPtr->showScrollBars = SHOW_VERTICAL_SCROLL_BAR;
    portalPtr->showScrollBars |= TEXT_SENSITIVE_SCROLL_BARS;
    portalPtr->verticalScroll = SCROLL_ON;

```

```

/* * Start filling in the static portal lines.
*/

line = 0;
NWSprintf(string, MSG("Serial Number      : ", 496));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Site ID          : ", 497));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Have Keys Status : ", 498));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Number Of Communities : ", 499));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Communities : ", 500));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

GblDataCol = BORDER_WIDTH + 24;

UpdateAdapterInformation(BackgroundPortal);
BackgroundFuncPtr = UpdateAdapterInformation;
HandleScrollablePortal(portalPtr);
BackgroundFuncPtr = NULL;
NWSDestroyPortal( BackgroundPortal, NUTHandle );
NWSSelectPortal( oldPortal, NUTHandle );
}

void DisplayMLIDStats(void)
{
    int     line;
    int     count;
    int     numberOfGenerics;
    int     len;
    int     promptMax;
    struct DriverStatsStructure *stats;
    struct DriverConfigurationStructure *config;
    ProtocolNodeStructure *protocol;
    CustVars *customStrings, *ptr;
    BYTE    oldPortal;
    BYTE    name[128], *namePtr;
    PCB     *portalPtr;
    BYTE    string[80];

    GblDataCol = ( ScreenWidth - 4 ) - BORDER_WIDTH - STATS_DATA_WIDTH;
    if ( DPCGetMLIDStats(&stats) )
    {
        return;
    }

    DIOGetMLIDConfig(&config);

/* * Build up the LAN name followed by its I/O resources

```

```

* to be used as the portals header.
*/

```

```

    if (config->DLogicalName[0] != 0)
        NWSprintf(name, MSG("%s %s", 270), config->DLogicalName, config
->DShortName);
    else
        NWSprintf(name, MSG("%s", 271), config->DShortName);

    if (config->DIOPortsAndRanges[1] > 0)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" port=%X", 272), config->DIOPortsAndRang
es[0]);
    }
    if (config->DIOPortsAndRanges[3] > 0)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" port=%X", 273), config->DIOPortsAndRang
es[2]);
    }
    if (config->DMemoryDecodeAndLength[0].LANMemoryAddress > 0 )
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" memory=%X", 274),
            config->DMemoryDecodeAndLength[0].LANMemoryAddre
ss);
    }
    if (config->DMemoryDecodeAndLength[1].LANMemoryAddress > 0 )
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" memory=%X", 275),
            config->DMemoryDecodeAndLength[1].LANMemoryAddre
ss);
    }
    if (config->DIntLine[0] != 0xff)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" int=%X", 276), config->DIntLine[0]);
    }
    if (config->DIntLine[1] != 0xff)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" int=%X", 277), config->DIntLine[1]);
    }
    if (config->DDMALine[0] != 0xff)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" dma=%X", 278), config->DDMALine[0]);
    }
    if (config->DDMALine[1] != 0xff)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" dma=%X", 279), config->DDMALine[1]);
    }
    if (config->DMediaType != NULL)
    {
        namePtr = name + CStrLen(name);
        NWSprintf(namePtr, MSG(" frame=%s", 280), config->DMediaType);
    }
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(""), 281);

```

```

/*
* Before we can create the portal, we must add up the number
* of lines we'll need, which will be bigger than the window.

```

```

    line = 3;
    protocol header */

    /* lines for version, node address, and

/* Add up the number of protocols bound to this board */
    protocol = MLIIDProtocolListByBoard( DIOBoard );
    while (protocol != NULL)
    {
        ++line;
        protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
    }

    /* Add in the number of generic statistics */
    line += 2; /* Blank line and Generic statistics header */

    numberOfGenerics = stats->GenericVariableCount;
    line += numberOfGenerics;

    promptMax = 0;
    for (count = 0; count < numberOfGenerics ; count++)
    {
        len = CStrLen( GetMsg(GenericDescriptionTable[count]) );
        if (promptMax < len)
        {
            promptMax = len;
        }
    }

    /* Add in the number of custom statistics */
    line += 2; /* Blank line and Custom statistics header */

    customPtr = (CustVars *)(&stats->CustomVariableCount);
    customStrings = (BYTE *) (customPtr->CustomVariable[customPtr->CustomVari
ableCount]);
    customStrings += sizeof(WORD);

    line += customPtr->CustomVariableCount;

    /* Check lengths of custom strings */

    ptr = customStrings; /* temp pointer to walk down custom prompts */
    for ( count = 0; count < customPtr->CustomVariableCount; count++)
    {
        len = CStrLen( ptr );
        if ( promptMax < len )
            promptMax = len;
        while ( *( ptr++ ) )
            ; /* find the next string */
    }

    line += 1; /* add a blank line for the bottom */

    if ( promptMax < 46 )
        promptMax = 64; /* minimum portal width */
    else if ( promptMax > 60 )
        promptMax = 76; /* maximum portal width */
    else
        promptMax += 16; /* custom portal width within range */

    /* build the portal */

```

```

if ( line < 8 )
    line = 8;
/* must meet the minimum portal size */

oldPortal = NUTHandle->currentPortal;
NWSDeselectPortal( NUTHandle );
BackgroundPortal = NWSCreatePortal
(
    1 /* gblUPFTopLine */,
    1 /* gblUPFLeftCol */,
    ScreenHeight - 3 /* gblUPFHeight + gblLPFHeight */,
    ScreenWidth - 2 /* gblUPFWidth */,
    line,
    ( ScreenWidth - 4 /* gblUPFWidth - 2 */ ),
    TRUE,
    VNORMAL,
    SINGLE,
    VINTENSE,
    CURSOR_OFF,
    VIRTUAL,
    NUTHandle
);
if ( BackgroundPortal > MAXPORTALS )
{
    NWSDeselectPortal( oldPortal, NUTHandle );
    return;
}

NWMSGetPCB( &portalPtr, BackgroundPortal, NUTHandle );
portalPtr->showScrollBar = SHOW_VERTICAL_SCROLL_BAR;
portalPtr->showScrollBar |= TEXT_SENSITIVE_SCROLL_BARS;
portalPtr->verticalScroll = SCROLL_ON;
NWSClearPortal( portalPtr );

/* Start filling in the static portal lines.
 */

/* Fill in the MLID version */

line = 0;
NWSPrintf(string, MSG("Version %d.%d", 282), config->DMLID_MajorVersion,
config->DMLID_MinorVersion);
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

/* Fill in the node address */

NWSPrintf(string, MSG("Node Address: %X%4.4X", 283),
    GET_HILO_LONG( (LONG *)&config->DNodeAddress[0]),
    GET_HILO_WORD( (WORD *)&config->DNodeAddress[4]));
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

/* protocols */

NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Protocols:", 284),
    CStrLen( MSG("Protocols:", 287) ), portalPtr );
protocol = MLIDProtocolListByBoard( DIOBoard );
while ( protocol != NULL )
{
    GetProtocolNameTableEntry( protocol->ProtocolNumber, string );
    CMovB(ProtocolNameTable[protocol->ProtocolNumber], string, 16);
    len = string[ 0 ];
    string[ len + 1 ] = NULL;
    NWSShowPortalLine

```

```

        line++,
        BORDER_WIDTH + INDENT_WIDTH,
        &string[ 1 ],
        len,
        portalPtr
    );
    if ( LStrCmp( string, MSG("\x003IPX", 285) ) == 0 )
    {
        LONG tmpLong;
        /* network address */

        tmpLong = IPXNetNumberTable[ DIOBoard ];
        NWSPrintf( string, MSG("Network address: ", 286), GET_HILO_LONG( &tmpLong ));

        len = CStrLen( string );
        NWSShowPortalLine
        (
            line++,
            BORDER_WIDTH + ( 2 * INDENT_WIDTH ),
            string,
            len,
            portalPtr
        );

        protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
    }
    /* Generic Statistics */
    line++;
    len = CStrLen( MSG("Generic statistics", 288) );
    NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Generic statistics", 289),
    len, portalPtr );
    GenericLineStart = line;
    promptMax -= 16;
    for ( count = 0; count < numberOfGenerics; count++ )
    {
        ptr = GetMsg(GenericDescriptionTable[ count ]);
        len = CStrLen( ptr );
        if ( len > promptMax )
            len = promptMax;
        NWSShowPortalLine
        (
            line++,
            BORDER_WIDTH + INDENT_WIDTH,
            ptr,
            len,
            portalPtr
        );

        /* custom stats */
        line++;
        len = CStrLen( MSG("Custom statistics", 227) );
        NWSShowPortalLine( line++, BORDER_WIDTH, MSG("Custom statistics", 292),
        len, portalPtr );

        for ( count = 0; count < customPtr->CustomVariableCount; count++ )
        {
            NWSShowPortalLine
            (
                line++,

```



```

    oint);
    {
        CLSLGetMLIDControlEntry(board, (void(*)())&ControlEntryP
        if (ControlEntryPoint)
        {
            config = (struct DriverConfigurationStructure *)
                CommandMLID(board
d, 0, (LONG)ControlEntryPoint); if (config)
    {
        configName = config->DShortName;
        if (!CStrCmp(configName, DPCName))
            goto FoundDPCBoardNumber;
    }
    delay(500);
    if (NWSKeyStatus(NUTHandle))
    {
        NWSGetKey(&type, &value, NUTHandle);
        if ((type == K_ESCAPE) || (type == K_AF10))
        {
            NWSEndWait(NUTHandle);
            return;
        }
        if (--countdown == 0)
        {
            Spin(NUTHandle);
            countdown = MAX_COUNTDOWN;
        }
    }
    /*
     * OK. We have a DPC MLID. Lets set up the main menu and
     * wait for the user to do something.
     */
    #endif

    #if !DRIVER_IO
    FoundDPCBoardNumber:
    #endif

    NWSEndWait(NUTHandle);
    NWSDisableInterruptKey(K_AF10, ExitHandler, NUTHandle);

    /*
     * Initialize the main options menu.
     */

    NWSInitDhList(NUTHandle, Free); /* Don't sort menu items. */

    NWSSetDynamicMessage(DYNAMIC_MESSAGE_ONE,
        (BYTE *) "Package Delivery ", &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_TWO,
        MSG("Display MLID Stats ", 102), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_THREE,
        MSG("DPC Configuration ", 95), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_FOUR,
        MSG("Dish Pointing ", 344), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_FIVE,
        MSG("Adapter Information ", 150), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_SIX,
        MSG("Modem Control ", 501), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_SEVEN,

```

```

        MSG("Exit DPCAGENT ", 586), &NUTHandle->messages);
    if (PackageDelivery)
        NWSAppendToMenu(DYNAMIC_MESSAGE_ONE, 1, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_TWO, 2, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_THREE, 3, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_FOUR, 4, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_FIVE, 5, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_SIX, 6, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_SEVEN, 7, NUTHandle);
    while (exitFlag == FALSE)
    {
        prevChoice = choice;

        /* Set defaultList to previous choice */
        defaultList = NWSGetListHead(NUTHandle);
        if (!PackageDelivery)
            choice = 1;
        for( i = choice - 1; i; i--)
            defaultList = defaultList->next;

        choice = NWSMenu(InXMSG("DPCAGENT Options", 151),
            10, 40, defaultList, NULL, NUTHandle, NULL);
        if (removedCount != DIORemovedCount)
        {
            NWSDestroyMenu(NUTHandle);
            NWSClearPortal( NUTHandle->portal[mainPortal] );
            goto LookForAdapter;
        }
        switch (choice) {
        case 1:
            if (NWSPushList(NUTHandle)) {
                DisplayPDInterface();
                NWSPopList(NUTHandle);
            }
            break;
        case 2:
            /* Display MLID Stats */
            if (NWSPushList(NUTHandle)) {
                DisplayMLIDStats();
                NWSPopList(NUTHandle);
            }
            break;
        case 3:
            /* Modem Configuration */
            if (NWSPushList(NUTHandle)) {
                DPCConfiguration();
                NWSPopList(NUTHandle);
            }
            break;
        case 4:
            /* Signal Strength Meter */
            if (NWSPushList(NUTHandle)) {
                DPCPointing();
                NWSPopList(NUTHandle);
            }
            break;
        case 5:
            /* Display Adapter Information */
            if (NWSPushList(NUTHandle)) {
                DisplayAdapterInfo();
                NWSPopList(NUTHandle);
            }

```

```

)
break;

case 6:
/* Display Adapter Information */
if (NWSPushList(NUTHandle)) {
ModemControl();
NWSPopList(NUTHandle);
}
break;

default:
if (NWSConfirm(InxMSG("Exit DPCAGENT?", 152), 0, 0, TRUE, NULL
    NUTHandle, NULL) == TRUE)
    exitFlag = TRUE;
else
    if (choice != 7)
        choice = prevChoice;
}

NWSDestroyMenu(NUTHandle);
}

/*****
*
* DPCAgentMain(void *parm)
*
* Description:
* Main thread. It will initialize the NUT screen and wait
* for user input.
*
* Input:    parm
*           - ignored
*
* Output:   Nothing
*
* Returns:  Nothing
*
*****/
void DPCAgentMain(void *parm)
{
    parm = parm;

    MainOptionsHandler();

    ReturnResources(1);
    exit(1);
}

/*****
*
* main(int argc, char *argv[])
*
* Description:
* Initialization routine.
*
* Input:    Nothing
*
*****/

```

```

*
* Output:   Nothing
*
* Returns:  0 if successfully initialized
*
*****/
LONG ScreenID;
LONG main(int argc, char *argv[])
{
    LONG currentScreen;
    LONG ser[2];
    LONG ccode;
    int i;

    for (i = 1; i < argc; i++)
    {
        if (ICmpB(argv[i], MSG("-DEBUG", 182), 6) == -1)
        {
            if ((DebugFlag = strtol(&argv[i][6], 0, 16)) == 0)
                DebugFlag = TRUE;
        }
    }

    /* Get a handle for allocating a resource tag */
    NLMHandle = (struct LoadDefinitionStructure *)GetNLMHandle();
    if (!NLMHandle)
        return(-1);

    if (!ReturnMessageInformation((LONG)NLMHandle, (BYTE ***)&NLMMessageTabl
e, NULL, NULL, NULL))
        return(-1);

    OSGetCountryInfo(&GblDOSCountryInfo);

    /* Allocate a resource tag to use for memory allocations */
    allocRTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Memory", 167
    ),
    AllocSignature);

    AESTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT AES", 168),
    AESProcessSignature);

    asyncIOtag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Async I/O",
    169),
    ASYNCSIOSignature);

    timerTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Delay Timer",
    170),
    TimerSignature);

    if (allocRTag == NULL ||
        AESTag == NULL ||
        timerTag == NULL ||
        ASYNCSIOSignature == NULL)
    {
        ConsolePrintf(TxtMSG("DPCAGENT: Unable to allocate Resource Tags
        ", 171));
        return(-1);
    }

    /* Create a screen for displaying our information */

```

```

currentScreen = GetCurrentScreen();
SetAutoScreenDestructionMode(TRUE);

/* Initialize the screen interface */
ScreenID = CreateScreen("DPCAGENT Utility", AUTO_DESTROY_SCREEN);
ccode = NWSInitializeNut(InxMSG("DPC AGENT PROGRAM", 172), AGENT_VERSION);

    SMALL_HEADER, NUT_REVISION_LEVEL, NULL, NULL,
    ScreenID, (LONG)allocrTag, &NUTHandle);
    if (ccode)
    {
        ConsolePrintf(TxtMSG("DPCAGENT: Unable to initialize NUT.", 174))
        return(-1);
    }
}
NLMMessageTable = (BYTE **)&(NUTHandle->messages);

/* Get a connection with the server we're on */
if (DebugFlag) {
    DisplayScreen(ScreenID);
    SetCurrentScreen(currentScreen);
}
#ifdef LOG_ECB_ACTIVITY
    if (DebugFlag >= 0x51) {
        DPC_TGID = GetThreadGroupID();
        LogRegisterClient("SYS:DIRECPC/LOG.CFG", 2048, &LogClientHandle);
        LogRegisterEvent("ECB", &LogECBHandle);
    }
#endif /* LOG_ECB_ACTIVITY */
}
else {
    DestroyScreen(currentScreen);
}
}
if (DEBUG_ALL
nID))
{
    ConsolePrintf(MSG("DPCAGENT: Unable to create debug screen.", 22
5));
    return(-1);
}
#endif

DPCUpdateConfig();

InetChangeProtocol();

DPCSetMaxConnections(ser);

DPCAgentPID = BeginThread(DPCAgentMain, NULL, NULL, NULL);
RenameThread(DPCAgentPID, "DPCAgent Main");
if (PackageDelivery) {
    DPCFilePID = BeginThread(DPCFileMain, NULL, 32 * 1024, NULL);
    RenameThread(DPCFilePID, "DPCAgent PD");
    DPCAccessPID = BeginThread(AccessMain, NULL, NULL, NULL);
    RenameThread(DPCAccessPID, "DPCAgent Access");
}
DPCModemPID = BeginThread(DloMain, NULL, NULL, NULL);
RenameThread(DPCModemPID, "DPCAgent Modem");
if (DPCMaxConnections) {
    DPCInetPID = BeginThread(InetMain, NULL, NULL, NULL);
    RenameThread(DPCInetPID, "DPCAgent Tinet");
}
signal(SIGTERM, ReturnResources);
ExitThread(TSR_THREAD, 0);
}

```

```

ReturnResources(int sig)

Description:      Shutdown routine. Returns the modules resources.

Input:      sig
              - ignored

Output:      Nothing

Returns:      Nothing

```

```
void ReturnResources(int sig)
{

```

```
    int i;
    int countdown = MAX_COUNTDOWN;

```

```
    sig = sig;
    ExitingFlag = TRUE;

```

```
    if (InReturnResources)
        return;

```

```
    InReturnResources = TRUE;

```

```

/*
 * Force NUT to escape out of all menus so that it can clean
 * before we call NWSRestoreNUT(NUT has a bug were it will
 * attempt to free up its memory twice(ABEND) if the user
 * leaves the screen a couple of menus in before unloading
 * the application from the command line.
 */

```

```
for(i = 0; i < 4; i++)
    NWSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

```

```
StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle,
    InxMSG("Waiting for threads to Exit", 226));

```

```
if (AccessAsleep)
    ResumeThread(DPCAccessPID);

```

```
if (InetAsleep)
    ResumeThread(DPCInetPID);

```

```

/*
 * Give threads and NUT a chance to execute.
 * Wait 1.5 seconds since signal meter and stats could be sleeping
 * for up to a second.
 */

```

```
delay(1500);

```

```

while(DPCFilePID || DPCModemPID || DPCAccessPID || DPCInetPID) {
    void DPCPDTerminate(void);
    DPCPDTerminate();
    if (AccessAsleep)
        ResumeThread(DPCAccessPID);
}

```

Thu Jul 17 14:46:11 1997

dpcagent.c

Page 51

03727800

```
if (InetAsleep)
    ResumeThread(DPCInetPID);

if (--countdown == 0) {
    Spin(NUTHandle);
    countdown = MAX_COUNTDOWN;
}
ThreadSwitchWithDelay();
}

#ifdef LOG_ECB_ACTIVITY
if (DebugFlag >= 0x51) {
    LogDeregisterEvent(&LogECBHandle);
    LogDeregisterClient(&LogClientHandle);
}
#endif /* LOG_ECB_ACTIVITY */

if (DlCfg.out_protocol == OUT_PPP)
{
    DisconnectPPP();
}
DIODERegisterAgent();

#ifdef DEBUG_ALL
CloseScreen(DebugScreenID);
#endif
NWSEndWait(NUTHandle);
NWSRestoreNut(NUTHandle);
}
```

```

#include "dpcagent.h"
/* Our header file */

LONG
int
/*
 * Access Configuration Variables
 */
BYTE SiteID[9];
WORD CDBVersion = 0;
WORD CDBethVersion = 0;
BYTE DacauFlag = 0;
LONG DacauTime = 0;
LONG RndDacauTime = 0;
DACAurequest_t DacauRequest;
CASDBbuffer CASDBpacau;
CASDBbuffer CASDBdcau;
CASDBbuffer CASDBbecau;

/* Elements tables */
static CDBelement_t Elements[MAXELEMENTS];
static CDBelement_t UpdatedElements[MAXELEMENTS];

BYTE *RcvBuf;
MACrecord_t CASmacs[MAX_MAC_RECORDS];
BYTE AccessAddress[] = {0x03, 0x00, 0x00, 0x00, 0x00, 0x00};
LONG AccessAsleep = FALSE;
/* Access thread needs t
o wake up flag */

BYTE
/* Stores current packet
 */
/* ECB linked list varaibles.
 */
static ECB *AccessECBHead = 0; /* Take ECBs from here */
static ECB *AccessECBTail = 0; /* Put ECBs here */

/*
 * element key used by LroSetAddress.
 */
BYTE MagicKey[] = { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 };

/*
 * Address Type Table used by MACbuildAddr().
 */
static unsigned char table[5][2] = {
    { MAC_INDIVID, MAC_NORMAL },
    { MAC_INDIVID, MAC_BYPASS },
    { MAC_MULTICAST, MAC_NORMAL },
    { MAC_MULTICAST, MAC_BYPASS }
};

BYTE SerialNum[9];
BYTE SerialNumpacked[3];

/*
 * ReadConfig(void)
 */
/*
 * Description: This routine reads the DPC.CFG file and stores the conte
nts
 */

```

into the appropriate globale variables.

Input: Nothing
Output: Nothing
Returns: Nothing

static void ReadConfig(void)

```

{
    int handle, k;
    BYTE *mem_ptr;

    for(k = 0; k < MAXELEMENTS; k++)
    {
        Elements[k].in_use = 'N';
        UpdatedElements[k].in_use = 'N';
    }
}

```

```

handle = open(MSG("SYS:DIRECPC\\DB\\DPC.CFG", 203), O_RDONLY);
if (handle != -1)
{

```

```

    read(handle, &SiteID, sizeof(SiteID));
    read(handle, &CDBVersion, sizeof(CDBVersion));
    read(handle, &CDBethVersion, sizeof(CDBethVersion));
    read(handle, &DacauFlag, sizeof(DacauFlag));
    read(handle, &DacauTime, sizeof(DacauTime));
    read(handle, &DacauRequest, sizeof(DACAurequest_t));
    read(handle, &CASDBpacau, sizeof(CASDBbuffer));
    read(handle, &CASDBbepb, sizeof(CASDBbuffer));
    read(handle, &CASDBdcau, sizeof(CASDBbuffer));
    read(handle, &CASDBbecau, sizeof(CASDBbuffer));
}

```

```

else
    UpdateFileStatus(MSG("Obtaining Encryption Keys", 204));

```

```

for(k = 0; k < MAX_MAC_RECORDS; k++)
    CASmacs[k].in_use = 'N';

```

```

if(handle != -1)
{
    close(handle);
}
else
{

```

```

    SiteID[0] = '\0';
    CASDBpacau.version = CASDBbepb.version = 0;
    CASDBdcau.version = CASDBbecau.version = 0;
    CASDBpacau.entries = CASDBbepb.entries = 0;
    CASDBdcau.entries = CASDBbecau.entries = 0;
}

```

```

CASDBpacau.entry_len = PACAU_LEN;
CASDBbepb.entry_len = PEB_LEN;
CASDBdcau.entry_len = DACAU_LEN;
CASDBbecau.entry_len = ECAU_LEN;

```

```

if (DacauFlag)
{

```

```

    UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 257));
    WaitingForKeys = TRUE;
}

```

```
DacauRequest.opcode = DACAU_REQUEST;
RndDacauTime = time(0) + rand();
```

```
★ SaveConfig(void *parm)
```

```

*
* Description:      This routine writes the access structures out to DPC.CFG
*
*                  a change is made to any of the structures
*
*                  anytime

```

★ Input:

* Output:

★ Returns:

* * *

```
static void SaveConfig(void)
```

```
int handle;  
int tmpFlag;
```

```

IWRITE | S_READ);
handle = open(MSG("SYS:DIREPC\\DB\\DPC.CFG", 206), O_RDWR | O_CREAT, S_

```

```
tmpFlag = DacauFlag;
if (WaitingForKeys)
```

DacauFlag = 1;

```

/* Write version */
write(handle, &cbv, sizeof(cbv));
write(handle, &cbvVersion, sizeof(cbvVersion));
write(handle, &cbvEtag, sizeof(cbvEtag));
write(handle, &cbvFlag, sizeof(cbvFlag));
write(handle, &cbvTime, sizeof(cbvTime));
write(handle, &cbvRequest, sizeof(cbvRequest));

```

```

/* Write Info headers */
write(handle, &CASDBpacau, sizeof(CASDBbuffer));
write(handle, &CASDBpeb, sizeof(CASDBbuffer));
write(handle, &CASDBdacau, sizeof(CASDBbuffer));
write(handle, &CASDBbecau, sizeof(CASDBbuffer));

```

```
/* Write buffers */
close(handle);
```

DacauFlag = tmpFlag;

* AccessESR(ECB *ecb)

* Description:

* This routine is called by the MLID interrupt service
* routine when a packet is received. The ECB is queued
* and if the Access File thread is sleeping, it is

★ woken up.

```

*      Input:          ecb
*
the packet

```

```

*      *
|      |
Output: nothing

```

* Returns:

```
* * * 0 we always keep the ECB
*****
```

```
int AccessESR(ECB *ecb)
```

```

/* Link the ECB to the Tail of the linked list */
ecb->ECB_NextLink = 0;
if (AccessECBTail)
    AccessECBTail->ECB_NextLink = ecb;
AccessECBTail = ecb;
if (AccessECBHead == 0)
    AccessECBHead = ecb;

```

```

/* Wake up file thread only if we need to */
if (AccessAsleep)
    ResumeThread(DPCAccessPID);

```

```
#ifdef LOG_ECB_ACTIVITY
    if (LogECBHandle) {
        int TGID = SetThreadGroupID(DPC_TGID);
        LogMsg(LogClientHandle, LogECBHandle, FALSE,
                "ACCESS queue(%08lx) %n", ecb);
        SetThreadGroupID(TGID);
    }
#endif
```

```
#endif /* LOG_ECB_ACTIVITY */
```

```
return(0);
```

```

/******
 *
 *      find_peb(ID element_pattern,
 *              char ver pattern)
 *
 */

```

* * * *	Description:
* * * *	This routine searches the PEB list to find an entry that matches the element and version pattern passed in.

```

* Input:
* element_pattern - 3 byte element to search for
* ver_pattern     - 1 byte version
*
to search for

```

★ **Output:**

```

*
*
*
*
Returns:
    pointer to Peb entry if it was found
    otherwise its a NULL

```

```

- MUXecau_t *find_ecau(ID group_pattern, char ver_pattern)
- {

```

```

int i;
MUXecau_t *p_ecau, *ret = NULL;
for(i = 0; i < CASDBecau.length; i += ECAU_LEN)
{
    p_ecau = (MUXecau_t *) (CASDBecau.p_buffer + i);
    if(CCmpB(&p_ecau->groupid, &group_pattern, sizeof(ID)) == -1)
    {
        if(ver_pattern == -1)
        {
            ret = p_ecau;
            break;
        }
        else
        {
            if(ver_pattern == p_ecau->version)
            {
                ret = p_ecau;
                break;
            }
        }
    }
    return(ret);
}

static MUXpeb_t *find_peb(ID element_pattern, char ver_pattern)
{
    unsigned short i, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;
    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        if(CCmpB(&p_peb->elementid, &element_pattern, sizeof(ID)) == -1)
        {
            if(ver_pattern == -1)
            {
                ret = p_peb;
                break;
            }
            else
            {
                if(ver_pattern == p_peb->version)
                {
                    ret = p_peb;
                    break;
                }
            }
        }
        num_addr = p_peb->num_addr[0];
        num_addr |= ((unsigned short)p_peb->num_addr[1]) << 8;
        i += num_addr * MAC_LENGTH;
    }
    return(ret);
}

/*
 * Deletes element not only from the CASDB
 * but from adapter as well
 */
/*****
static del_peb_element(int e_num)
{
    int k;

```

```

*
* Description: This routine deletes and from the CASDB and from the
* adapter.
*
* Input: e_num - index of the element t
*
* Output: nothing
*
* Returns: 0
*
*****
static del_peb_element(int e_num)
{
    int k;
    DIDeleteAddress (Elements[e_num].channel, (BYTE *) &Elements[e_num].e_mac);
    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
           CCmpB(&CASmacs[k].dpc_mac, &Elements[e_num].e_mac, sizeof(MACAdd
           r_t)) == -1)
        {
            CASmacs[k].in_use = 'N';
            break;
        }
    }
    Elements[e_num].in_use = 'N';
    return(0);
}
/*****
*
* replace_peb_element(int num,
* unsigned char new_ver)
*
* Description: This routine replaces the version numbers of the element
* indexed by num.
*
* Input: int_num
* nt to modify new_ver
* stuff into Element entry
*
* Output: nothing
*
* Returns: 0
*
*****
static replace_peb_element(int num, unsigned char new_ver)
{
    int k;

```

```

for(k = 0; k < MAX_MAC_RECORDS; k++)
{
    if(CASmacs[k].in_use == 'Y' &&
        CCmpB(&CASmacs[k].dpc_mac, &Elements[num].e_mac, sizeof(MACAddr_
t)) == -1)
    {
        CASmacs[k].dpc_mac.Ver = new_ver;
        break;
    }
    Elements[num].e_mac.Ver = new_ver;
    Elements[num].e_ver = new_ver;
    return 0;
}

/*****
*
* find_pacau(ID group_pattern,
* char ver_pattern)
*
* Description: This routine attempts to locate a pacau given a group
* pattern.
*
* Input: group_pattern
* ver_pattern
*
* Output:
*
* Returns: pointer to pacau if successful
* Otherwise NULL
*
*****/
MUXpacau_t *find_pacau(ID group_pattern, char ver_pattern)
{
    int i;
    MUXpacau_t *p_pacau, *ret = NULL;
    for(i = 0; i < CASDBpacau.length; i += PACAU_LEN)
    {
        p_pacau = (MUXpacau_t *)CASDBpacau.p_buffer + i;
        if(CCmpB(&p_pacau->groupid, &group_pattern, 3)) == -1)
        {
            if(ver_pattern == -1)
            {
                break;
            }
            else
            {
                if(ver_pattern == p_pacau->version)
                {
                    ret = p_pacau;
                    break;
                }
            }
        }
        return(ret);
    }
}

/*****
*
* reverse_key(BYTE *key)
*
* Description: This routine swaps the byte order of the 8 byte
* key passed in.
*
* Input: key
*
* Output:
*
* Returns: pointer to pacau if successful
* Otherwise NULL
*
*****/

```

```

*
* find_dacau(ID group_pattern,
* char ver_pattern)
*
* Description: This routine attempts to locate a dacau given a group
* pattern.
*
* Input: group_pattern
* ver_pattern
*
* Output:
*
* Returns: pointer to dacau if successful
* Otherwise NULL
*
*****/
MUXdacau_t *find_dacau(ID group_pattern, char ver_pattern)
{
    int i;
    MUXdacau_t *p_dacau, *ret = NULL;
    for(i = 0; i < CASDBdacau.length; i += DACAU_LEN)
    {
        p_dacau = (MUXdacau_t *)CASDBdacau.p_buffer + i;
        if(CCmpB(&p_dacau->groupid, &group_pattern, 3)) == -1)
        {
            if(ver_pattern == -1)
            {
                break;
            }
            else
            {
                if(ver_pattern == p_dacau->version)
                {
                    ret = p_dacau;
                    break;
                }
            }
        }
        return(ret);
    }
}

/*****
*
* reverse_key(BYTE *key)
*
* Description: This routine swaps the byte order of the 8 byte
* key passed in.
*
* Input: key
*
* Output:
*
* Returns: pointer to pacau if successful
* Otherwise NULL
*
*****/

```

```
void reverse_key(BYTE *key)
```

```
{
    unsigned char x;
```

```
    x = key[0]; key[0] = key[1]; key[1] = x;
    x = key[2]; key[2] = key[3]; key[3] = x;
    x = key[4]; key[4] = key[5]; key[5] = x;
    x = key[6]; key[6] = key[7]; key[7] = x;
}
```

```
/******
```

```
    make_element_id(BYTE *e_id,
                    char *e_id_txt)
```

```
    Description:    This routine is called by LroSetAddress to help
                    build the element id.
```

```
    Input:         e_id
```

```
    - element id
```

```
    Output:        e_id_txt
```

```
    - element text
```

```
    Returns:       nothing
```

```
/******
```

```
void make_element_id(BYTE *e_id, char *e_id_txt)
```

```
{
    BYTE work[3];
    static char HexChar[] = MSG("0123456789ABCDEF", 208);
```

```
    unsigned char Ch, *p;
    int count = 3, i = 0;
```

```
    work[0] = e_id[2];
    work[1] = e_id[1];
    work[2] = e_id[0];
```

```
    p = work;
    while (count--)
```

```
    {
        Ch = *p++;
        e_id_txt[i++] = HexChar[Ch>>4]; /* high nibble */
        e_id_txt[i++] = HexChar[Ch & 0xf]; /* low nibble */
    }
    e_id_txt[i] = '\0';
}
```

```
/******
```

```
    int43(LONG id, BYTE *array)
```

```
    Description:
```

```
        This routine is called by MACbuildAddr to convert
        an id to its 3 byte equivalent.
```

```
    Input:         id
```

```
    - id to convert
```

```
r to 3 byte array
```

```
    Output:        array is filled in
```

```
    Returns:       0 if successful
```

```
static int43(LONG id, BYTE *array)
```

```
{
    union {
        BYTE b[4];
        LONG w;
    } offset;
    int status = 0;

    offset.w = id;
    if (offset.b[3]==0 && ! (offset.b[2] & 0xc0)) {
        array[0] = offset.b[2];
        array[1] = offset.b[1];
        array[2] = offset.b[0];
    }
    else {
        array[0]=array[1]=array[2] = 0;
        status = -1;
    }
    return status;
}
```

```
/******
```

```
    MACbuildAddr(char *element_txt,
                 int feature,
                 BYTE ver,
                 MACaddr_t *address)
```

```
    Description:    This routine is called by LroSetAddress to build a
                    MAC address out of a file_id and community id.
```

```
    Input:          element_txt
                    feature
```

```
    AC_PKG, MAC_DF
```

```
    _BYPASS_MULTICAST
```

```
    ver
```

```
    community id
```

```
    address
```

```
    ing address
```

```
    Output:         address is filled in
```

```
    Returns:       0 if successful
```

```
/******
```

```
int MACbuildAddr(char *element_txt, int feature, BYTE ver, MACaddr_t *address)
```

```
{
    BYTE element[3];
    LONG i;
```

```
    - file id array
    - MAC_HI, MAC_CAS_IND, M
    or MAC
    - low byte of co
    - where store the result
```

```

int k, status = 0;

if(feature < 0 || feature > 5)
    return(-1);
/* Step one */
sscanf(element_txt, MSG("%lx", 137), &i);
if((status = int43(i, element)) != 0)
    return(status);
/* Step two */
for(k=0; k<3; k++)
{
    element[k] = element[k] << 2;
    element[k] |= ((k == 2) ? 0x00 : element[k+1]) >> 6;
}
/* Step three */
address->Element[0] = element[2];
address->Element[1] = element[1];
address->Element[2] = element[0];
/* Set Multicast/Individual */
if(table[feature][0] == MAC_MULTICAST)
    address->Element[0] |= MAC_MULTICAST;
/* Set Bypass/Normal */
if(table[feature][1] == MAC_BYPASS)
    address->Element[0] |= MAC_BYPASS;
/* Set application ID or Version number */
switch(feature)
{
    case MAC_HI:
        address->Ver = 0x02;
        break;
    case MAC_CAS_IND:
        address->Ver = 0x01;
        break;
    case MAC_BYPASS_MULTICAST:
        address->Ver = 0x00;
        break;
    default:
        address->Ver = ver;
        break;
}
/* Set reserved field */
address->Reserved[0] = address->Reserved[1] = 0x00;
if(feature == MAC_DF)
    address->Element[2] = 0xff;
return(status);
}

/*****
 *
 * find_peb_mac(MACAddr_t mac_pattern)
 *
 * Description: Finds the PEB entry which contains the mac address passed in.
 *
 * Input: mac_pattern - MAC address to search for
 *
 * Output: Nothing
 *
 * Returns: NULL if no entry found
 *          Otherwise its a pointer to the PEB entry
 */

```

```

static MUXpeb_t *find_peb_mac(MACAddr_t mac_pattern)
{
    unsigned short i, j, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;
    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        num_addr = p_peb->num_addr[0];
        num_addr |= ((unsigned short)p_peb->num_addr[1]) << 8;
        for(j = 0; j < num_addr; j++)
        {
            if(CCmpB(&mac_pattern,
                _LENGTH, MAC_LENGTH) == -1)
            {
                ret = p_peb;
                goto peb_mac_exit;
            }
            i += num_addr * MAC_LENGTH;
        }
        peb_mac_exit:
        return(ret);
    }
}

/*****
 *
 * find_element_id(ID id,
 *                 BYTE ver)
 *
 * Description: Find the elements index into the Element table.
 *
 * Input: id - 3 byte
 *        ver - 1 byte version
 *
 * Output: Nothing
 *
 * Returns: -1 if not found
 *          otherwise its the index
 *
 * *****/
static find_element_id(ID id, BYTE ver)
{
    int k;
    int ret = -1;
    for(k = 0; k < MAXELEMENTS; k++)
    {
        if(Elements[k].in_use == 'Y' &&
            CCmpB(&Elements[k].e_id, &id, sizeof(ID)) == -1 &&
            Elements[k].e_ver == ver)
        {
            ret = k;
            break;
        }
    }
}

```



```

static check_df_groups(void)
{
    int i;
    MUXpacau_t *pacau;
    MUXdacau_t *dacau;
    MUXpeb_t *p_peb;

    for(i = 0; i < MAXELEMENTS; i++) {
        if(Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
            continue;
        if((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
            continue;
        pacau = find_pacau(p_peb->groupid, p_peb->version);
        dacau = find_dacau(p_peb->groupid, p_peb->version);

        if(dacau == NULL && pacau == NULL) {
            /* We have no group for this element */
            del_peb_element(i);
        }
        return 0;
    }
}

/*****
 *
 * parse_pacau(BYTE *buf, WORD len)
 *
 * Description: Parse the PACAU packet. This is where we detect that we
 * must receive a new key via the modem(packet's d version
 * will not match CASDBdacau.version.
 *
 * Input: buf - pointer to the message receive
 * len - length of the message received
 *
 * Output: Nothing
 *
 * Returns: 0 if successfully parsed
 *
 *****/
static parse_pacau(BYTE *buf, WORD len)
{
    LONG curr_p_version;
    LONG curr_d_version;
    LONG curr_d_time;
    int ret = 0, k;

    if(strncmp(buf, SiteID, 8) != ESUCCESS) {
        strncpy(SiteID, buf, 8);
        SiteID[8] = 0;
        CDBVersion++;
        /* New Site ID - reset versions of PACAU, DACAU, ...*/
        CASDBpacau.version = CASDBpeb.version = 0;
        CASDBdacau.version = CASDBbecau.version = 0;
        SaveConfig();
    }

    curr_p_version =

```

```

buf[8] * 256UL +
buf[9] * 256UL +
buf[10] * 256UL * 256UL +
buf[11] * 256UL * 256UL * 256UL;

curr_d_version =
buf[12] * 256UL +
buf[13] * 256UL +
buf[14] * 256UL * 256UL +
buf[15] * 256UL * 256UL * 256UL;

curr_d_time =
buf[16] * 256UL +
buf[17] * 256UL +
buf[18] * 256UL * 256UL +
buf[19] * 256UL * 256UL * 256UL;

if(curr_d_version != CASDBdacau.version) {
    /* There are some changes in the DACAU staff */
    /* Build DACAU request */
    DacauFlag = 1;
    srand(time(0));
    DacauTime = curr_d_time;
    RndDacauTime = time(0) + rand();
    CDBVersion++;

    WaitingForKeys = TRUE;
    UpdateFileStatus(MSG("Obtaining Encryption Keys", 138));

    memcpy(&DacauRequest.d_version, buf + 8 + sizeof(VER), sizeof(VER));
    #ifdef USE_NEW_MIPS_CODE
    DacauRequest.d_version.i[3] |= 0x80;
    #endif

    CASDBdacau.version = curr_d_version;
    SaveConfig();
}

if(curr_p_version != CASDBpacau.version) {
    CDBVersion++;
    CASDBpacau.version = curr_p_version;
    CASDBpacau.length = len - PACAU_HEAD_LEN;
    memcpy(CASDBpacau.p_buffer,
        buf + PACAU_HEAD_LEN,
        CASDBpacau.length);
    CASDBpacau.entries = CASDBpacau.length / CASDBpacau.entry_len;
    check_df_groups();
    SaveConfig();
}
return ret;
}

/*****
 *
 * parse_ecau(BYTE *buf, WORD len)
 *
 * Description: Parse the ECAU packet. Replace the old entry by the
 * new one as long as version doesn't match CASDBbecau versi
 * on.
 *
 * Input: buf - pointer to the message receive
 * len - length of the message received
 *
 *****/

```

```

*
* Output: Nothing
*
* Returns: 0 if successfully parsed
*
*****
static parse_ecau(BYTE *buf, WORD len)
{
    LONG curr_e_version;
    int ret = 0;

    curr_e_version = buf[0] +
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +
        buf[3] * 256UL * 256UL * 256UL;

    if (curr_e_version != CASDBecau.version)
    {
        CDBVersion++;
        CASDBecau.version = curr_e_version;
        CMovB(buf + ECAU_HEAD_LEN, CASDBecau.p_buffer, len - ECAU_HEAD_LEN);
        CASDBecau.length = len - ECAU_HEAD_LEN;
        CASDBecau.entries = CASDBecau.length / CASDBecau.entry_len;
        SaveConfig();
    }
    return ret;
}

/*****
*
* parse_peb(BYTE *buf, WORD len)
*
* Description: Parse the PEB packet. Replace the old entry by the
*              new one. Check elements and add new addresses to the
*              MLID and delete old ones.
*
* Input: buf - pointer to the message receive
*        len - length of the message received
*
* Output: Nothing
*
* Returns: 0 if successfully parsed
*
*****
static parse_peb(BYTE *buf, WORD len)
{
    int i, k, macs, not_found;
    MUXpeb_t *p_peb;
    unsigned long curr_version;
    int ret = 0;
    MUXpacau_t *pacau;
    MUXdacau_t *dacau;

    curr_version = buf[0] +
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +

```

```

        buf[3] * 256UL * 256UL * 256UL;

    if (curr_version != CASDBpeb.version)
    {
        CDBVersion++;
        CASDBpeb.version = curr_version;
        CMovB(buf + PEB_HEAD_LEN, CASDBpeb.p_buffer, len - PEB_HEAD_LEN);
        CASDBpeb.entries = CASDBpeb.length / CASDBpeb.entry_len;
        /* Walk throught the elements table and check ... */
        /* ... if the element still in the PEB */
        for (i = 0; i < MAXELEMENTS; i++)
        {
            if (Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
                continue;
            if ((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
            {
                /* Element no longer valid: Delete. */
                del_peb_element(i);
                continue;
            }
            if (p_peb->version != Elements[i].e_ver)
            {
                /* We have found element but with different version: */
                /* It means, that we somehow miss key update */
                /* Replace inside adapter and in CDB. */
                /* Delete address */
                DIODeleteAddress(Elements[i].channel, (BYTE *)&E
                lements[i].e_mac);

                /* Replace element in the CDB */
                replace_peb_element(i, p_peb->version);
                /* Trying to Add address */
                /* At first find group for the element */
                pacau = find_pacau(p_peb->groupid, p_peb->version);
                dacau = find_dacau(p_peb->groupid, p_peb->versio
                n);

                if (dacau != NULL)
                {
                    pacau = (MUXpacau_t *)dacau;
                    if (pacau != NULL)
                    {
                        if (DIOAddGroupAddress(Elements[i].chann
                        el, (BYTE *)&Elements[i].e_mac,
                        (BYTE *)&pacau->g_key))
                        {
                            /* Find a group, Add */
                            channelCfg.CfgChannel = Elements[i].chan
                            nel;
                            channelCfg.CfgNumAddresses = 1;
                            CMovB(&Elements[i].e_mac, channelCfg.Cfg
                            Address, 8);
                            CMovB(&pacau->g_key, key, 8);
                            reverse_key(&key);
                            CMovB(key, channelCfg.CfgGroupkey, 8);
                            CMovB(&MagicKey, channelCfg.CfgElementke
                            y, 8);
                            ecb.ECB_StackID = MLID_ADD_ADDRESS;
                            ecb.ECB_Fragment[0].FragmentAddress = &c
                            hannelCfg;
                            if (!octMLID(FDBBoard, &ecb, FDBControl
                            Entry) != 0)
                                del_peb_element(i);

```

```

    else
    {
        /* Can't find group for this element */
        /* We are not subscribed on this group any more */
        /* Delete element. */
        del_peb_element(i);
    }

    /* Check Ethernet addresses within Element. */
    macs = not_found = 0;
    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
           CCompB(&CASmacs[k].dpc_mac, &Elements[i].e_mac, sizeof(MACAddr_t)) == -1)
        {
            macs++;
            if(find_peb_mac(CASmacs[k].e_mac) == NULL)
            {
                /* Somebody in the NOC has deleted Ethernet */
                /* address of this element... */
                not_found++;
                CASmacs[k].in_use = 'N';
            }
        }
        if(macs == not_found)
        {
            /* There are no ethernet addresses for the element */
            del_peb_element(i);
        }
    }

    SaveConfig();
}

return ret;
}

/*****
 *
 * parse_gup(BYTE *buf, WORD len)
 *
 * Description:   Parse the GUP packet. Add new entries to the PACAU buffer.
 *
 * Input:        buf
 *               - pointer to the message receive
 *               len
 *               - length of the message received
 *
 * Output:       Nothing
 *
 * Returns:      0 if successfully parsed
 *
 *****/
static parse_gup(BYTE *buf, WORD len)
{
    GUPid_t *p_gup_element;
    GUPhead_t *p_gup_head;
    MUXpacau_t *p_pacau_tmp;
    int i, curr_len = 0;

    if (index = CCompB(&p_gup_head->adapterstart, SerialNum, sizeof(ID));
        if (index == -1)
            start = 0;
        else
            start = p_gup_head->adapterstart.i[index] - SerialNum[i];
        index = CCompB(&p_gup_head->adapterend, SerialNum, sizeof(ID));
        if (index == -1)
            end = 0;
        else
            end = p_gup_head->adapterend.i[index] - SerialNum[i];

        if(start <= 0 && end >= 0)
        {
            CDBVersion++;
            for(i = 0; i < p_gup_head->entries && curr_len < len; i++, len -= GUP_LEN)
            {
                p_gup_element = (GUPid_t *) (buf + GUP_HEAD_LEN + i * GUP_LEN);
                if(CCompB(&p_gup_element->adapternum, SerialNum, sizeof(ID)) == -1)
                {
                    p_pacau_tmp = (MUXpacau_t *) (CASDBpacau.p_buffer + CASDBpacau.length);
                    CASDBpacau.length += PACAU_LEN;
                    CASDBpacau.entries++;
                    CMovB(&p_gup_head->groupid, &p_pacau_tmp->groupid, sizeof(ID));
                    p_pacau_tmp->version = p_gup_head->g_ver;
                    CMovB(&p_gup_element->g_key, &p_pacau_tmp->g_key, sizeof(chunk));
                    break;
                }
            }
        }
        return 0;
    }

    /*****
    *
    * update_peb(BYTE *buf, WORD len)
    *
    * Description:   Parse the UPDATE_PEB packet. Replace the old entry by the
    *               new one according to the Element ID.
    *
    * Input:        buf
    *               - pointer to the message receive
    *               len
    *               - length of the message received
    *
    * Output:       Nothing
    *
    * Returns:      0 if successfully parsed
    *
    *****/
    static update_peb(BYTE *buf, WORD len)
    {
        int m;
        MUXpacau_t *pacau = NULL;
        MUXdcau_t *dcau = NULL;
    }

```

```

MUXpeb_t *old_peb, *new_peb, *found;
int found_element;
short ret_code = 0;
unsigned long curr_version;

if (len != 2 * PEB_LEN + PEB_HEAD_LEN)
    return(0);
curr_version =
    buf[0] * 256UL +
    buf[1] * 256UL +
    buf[2] * 256UL +
    buf[3] * 256UL * 256UL;

if (curr_version == CASDBpeb.version)
    return 0;
CDBVersion++;
CASDBpeb.version = curr_version;

old_peb = (MUXpeb_t *) (buf + PEB_HEAD_LEN);
new_peb = (MUXpeb_t *) (buf + PEB_HEAD_LEN + PEB_LEN);
if ((found = find_peb(old_peb->elementid, old_peb->version)) == NULL)
    /* Nothing to replace - ERROR */
    return(0);
for (m = 0; m < MAXELEMENTS; m++)
{
    if (UpdatedElements[m].in_use == 'Y')
    {
        /* We have received next replace element command,
        * it means, that the previous element
        * has already been updated at the
        * DataFeed lan Gateway and
        * we can delete old address and keys
        * from adapter
        */
        ret_code = DIDeleteAddress(UpdatedElements[m].channel,
            (BYTE *)&UpdatedElements[m].e_mac);
        UpdatedElements[m].in_use = 'N';
    }
}
/* Has been the element loaded before? */
found_element = find_element_id(old_peb->elementid, old_peb->version);
if (found_element != -1)
{
    /* Yes. We are receiving this element now.
    * Let's keep old element's address
    * in the UpdatedElement table
    */
    CMovB(&Elements[found_element], &UpdatedElements[found_element],
        sizeof(CDBelement_t));
    /* Trying to find a group for the element
    */
    pacau = find_pacau(new_peb->groupid, new_peb->grversion);
    dacau = find_dacau(new_peb->groupid, new_peb->grversion);
    if (dacau != NULL)
        pacau = (MUXpacau_t *) dacau;
        if (pacau != NULL)
        {
            /* Put element in the adapter
            * After this operation in the adapter will be
            * both old and new element's addresses and keys
            */
        }
}
}

delay(120);
ret_code = DIAddGroupAddress(Elements[found_element].ch
annel,
    (BYTE *)&pacau->g_key);
else
    /* No group, Delete element */
    del_peb_element(found_element);
/* Change PEB database for this element */
if (!ret_code)
{
    replace_peb_element(found_element, new_peb->version);
    CMovB( (unsigned char *)new_peb, (unsigned char *)found, PEB_LEN - 2);
    return(ret_code);
}
/*****
* AccessReceive(char *message)
* Description: This routine checks to see if we've received any
* packets from the MLID. If we have, the data is copied
* from the ECB to the message and the ECB is returned to t
* he
* LSL.
* Input: message
* r to where to copy data
* Output: message and lroinfo filled in if successful
* Returns: 0 if a packet has been received
*****
LONG AccessReceive(char *message)
{
    ECB *ecb;

    /* Extract an ECB from the linked list if one exists */
    Disable();
    ecb = AccessECBHead;
    if (!ecb)
    {
        /* No ecb. Just return */
        Enable();
        return(-1);
    }
    AccessECBHead = ecb->ECB_NextLink;
    if (AccessECBHead == 0)
        AccessECBTail = 0;
    Enable();

    /* copy the data past the lroinfo to the message */
    CMovB(ecb->ECB_Fragment(0).FragmentAddress, message, ecb->ECB_Fragment(0)
        .FragmentLength);

    /* return the ecb to the LSL */
}

```

```

CLSLReturnRcvECB(ecb);
#endif LOG_ECB_ACTIVITY
FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
"ACCESS return(%08lx)\n", ecb));
#endif /* LOG_ECB_ACTIVITY */
return(0);

```

```

/*****

```

```

* AccessAdd(BYTE *message)

```

```

* Description:

```

```

This routine is called when a packet is received
and is responsible for dispatching it.

```

```

* Input:

```

```

message - packet data
lroinfo - Lro information

```

```

* Output:

```

```

Nothing

```

```

* Returns:

```

```

Nothing

```

```

*****/

```

```

int AccessAdd(BYTE *message) {

```

```

    IndPacket_t *p_packet;

```

```

    int packet_type;

```

```

    int status;

```

```

    p_packet = (IndPacket_t *)message;

```

```

    if(p_packet->address[3] == 0x01) /* CAS Individual address */

```

```

    switch(p_packet->payload[0]) {

```

```

        case SG_PACAU:

```

```

            packet_type = PACAU;

```

```

            break;

```

```

        case SG_ECAU:

```

```

            packet_type = ECAU;

```

```

            break;

```

```

        }
    else if(p_packet->address[0] == 0x03) { /* Bypass key */

```

```

        switch(p_packet->payload[0]) {

```

```

            case PEB_PACKET:

```

```

                packet_type = PEB;

```

```

                break;

```

```

            case GUP_PACKET:

```

```

                packet_type = GUP;

```

```

                break;

```

```

            case PEB_UPDATE_PACKET:

```

```

                packet_type = PEB_UPDATE;

```

```

                break;

```

```

            default:

```

```

                packet_type = UNKNOWN;

```

```

                break;

```

```

        }

```

```

    else

```

```

        packet_type = UNKNOWN;

```

```

    switch(packet_type) {

```

```

        case UNKNOWN:

```

```

            break;

```

```

    case PACAU:
        status = parse_pacau(p_packet->payload+1, p_packet->length-1);
        break;
    case ECAU:
        status = parse_eacau(p_packet->payload+1, p_packet->length-1);
        break;
    case PEB:
        status = parse_peb(p_packet->payload+1, p_packet->length-1);
        break;
    case GUP:
        status = parse_gup(p_packet->payload+1, p_packet->length-1);
        break;
    case PEB_UPDATE:
        status = update_peb(p_packet->payload+1, p_packet->length-1);
        break;
    }
    return(status);
}

```

```

/*****

```

```

* parse_dacau(BYTE *buf,

```

```

    int len)

```

```

* Description:

```

```

Parse the new DACAU from the message received by the mod
em.

```

```

* Replace the old DACAU buffer by the new one if the the v

```

```

* matches what we have in the CASDBdacau version.

```

```

* Input:

```

```

buf - pointer to the message receive

```

```

len - length of the message received

```

```

* Output:

```

```

Nothing

```

```

* Returns:

```

```

0 if successful

```

```

*****/

```

```

static int parse_dacau(BYTE *buf, int len)

```

```

{
    unsigned long curr_d_version;

```

```

    int ret = 0;

```

```

    curr_d_version =

```

```

        buf[0] +

```

```

        buf[1] * 256UL +

```

```

        buf[2] * 256UL * 256UL +

```

```

        buf[3] * 256UL * 256UL * 256UL;

```

```

    if(curr_d_version == CASDBdacau.version) {

```

```

        CDBVersion++;

```

```

        CASDBdacau.version = curr_d_version;

```

```

        CMovB(buf + DACAU_HEAD_LEN, CASDBdacau.p_buffer, len - DACAU_HEAD_LEN);

```

```

        CASDBdacau.length = len - DACAU_HEAD_LEN;

```

```

        CASDBdacau.entries = CASDBdacau.length / CASDBdacau.entry_len;

```

```

        check_df_groups();

```

```

    }
    else {

```

```

        ret = -1;

```

```

    }
}

```



```

    parm = parm;

/*
 * When MLID has been found, Open the conditional access channel.
 */

RegisterWithDriver:

    while (!ExitingFlag)
    (
        AccessAsleep = TRUE;
        delay(500);
        AccessAsleep = FALSE;

        if (DIOBoard != 0)
        (
            removedCount = DIORemovedCount;
            if (AccessChannel != -1 ||
                DIOOpenChannel(AccessAddress, AccessESR,
                    &AccessChannel) == 0)
            (
                DIOGetSN(SerialNum);
                sn = atol(SerialNum);
                NWSprintf(SerialNum, MSG("%061X", 213), sn);

                pack_mac_addr(SerialNumPacked, 3, SerialNum, 6);
                x = SerialNumPacked[0];
                SerialNumPacked[0] = SerialNumPacked[2];
                SerialNumPacked[2] = x;

                MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACAddr
                    _t *)address);

                if (DIOAddrAddress(AccessChannel, address) == 0)

                    channelCfg.CfgChannel = AccessChannel;
                    channelCfg.CfgNumAddresses = 1;
                    MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACAddr
                        _t *)channelCfg.CfgAddress);

                    ecb.ECB_StackID = MLID_ADD_ADDRESS;
                    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
                    if (IoctlMlId(FDBboard, &ecb, FDBControlEntry) =
                        = 0)
                        break;

                    )
                )

            if (ExitingFlag)
            (
                DPCAccessPID = 0;
                return;
            )

/*
 * Now lets open up the the DPC.CFG file.
 */

ReadConfig();

while (!ExitingFlag)
(

```

```

    ccode = AccessReceive(AccessMessage);
    if (ccode)
    (

```

```

        AccessAsleep = TRUE;
        SuspendThread(DPCAccessPID);
        AccessAsleep = FALSE;
        if (removedCount != DIORemovedCount)
            goto RegisterWithDriver;
        continue;
    )

```

```

        AccessAdd(AccessMessage);
        if (DacauFlag)
            GetDacau();
    )

```

```

        DIOCloseChannel(AccessChannel);
        DPCAccessPID = 0;
    )

```

```

#include "dpcagent.h" /* Our header file */

LONG DIOBoard = 0;
LONG DIORemovedCount = 0;
LONG DIOControlEntry = 0;
struct DriverStatsStructure* DIOStats = 0;

void DIORemove(void)
{
    int i;

    /* Invalidate the DriverIO board. Increment removed count so that
     * other threads can detect that the driver has changed, even if
     * DIOBoard gets filled in. The other threads can then re-register
     * with the new driver.
     */

    DIOStats = 0;
    DIOBoard = 0;
    DIORemovedCount++;

    /* Force the NWSNUT menus to exit so that DPCAGENT can go back
     * to searching for a new adapter before allowing user to choose
     * options.
     */

    for(i = 0; i < 4; i++)
        NWSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

    /*
     * Force sleeping threads to wake up so that they can detect that
     * the adapter has been unloaded(DIORemovedCount will be different
     * from their local copy of the last removed count). The threads
     * should then spin(sleep) periodically until a new adapter is
     * present, and then re-register with the adapter if they need to.
     */

    if (AccessAsleep)
        ResumeThread(DPCAccessPID);

    if (InetAsleep)
        ResumeThread(DPCInetPID);

}

LONG DIORRegisterWithAdapter(char *shortName)
{
    LONG board;

    for(board = 0; board < NumberOfLANs; board++)
    {
        void (*ControlEntryPoint) (void) = NULL;
        if (CLSLGetMLIDControlEntry(board,
                                     &ControlEntryPoint) == ESUCCESS)
        {
            struct DriverConfigurationStructure* config = 0;
            if ((config = (struct DriverConfigurationStructure *) Co
mmmandMLid(board, 0, (LONG)ControlEntryPoint)))
            {
                if (!CStrCmp(config->DShortName, shortName))
                {
                    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC",
505));

```

```

void (*removeRoutine) () = DIORemove;

```

```

ecb.ECB_StackID = MLID_REGISTER_AGENT;
ecb.ECB_Fragment[0].FragmentAddress = &r

```

```

/* Call MLID */
IoctlMLid(board, &ecb, (LONG)ControlEntr

```

```

DIOBoard = board;
DIOControlEntry = (LONG)ControlEntryPoin

```

```

return(0);

```

```

    }
    return(-1);
}

```

```

void DIORRegisterAgent(void)
{

```

```

    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 506));
    void (*nullRoutine) () = 0;

```

```

    if (DIOBoard)
    {

```

```

        ecb.ECB_StackID = MLID_REGISTER_AGENT;
        ecb.ECB_Fragment[0].FragmentAddress = &nullRoutine;

```

```

        /* Call MLID */
        IoctlMLid(DIOBoard, &ecb, DIOControlEntry);
        DIOBoard = 0;
    }
}

```

```

/*****

```

```

 * DIOGetSN(char *serialNum)

```

```

 * Description:

```

```

 * This routine Gets the serial number from the adapter.
 * It is used for explicit requests of packages that are
 * for sale.

```

```

 * Input: serialNum - Pointer to where to store seri
al number

```

```

 * Output: serialNum - filled out if successful

```

```

 * Returns: 0 if successful

```

```

 * LONG DIOGetSN(char *serialNum)
{

```

```

    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 108));

```

```

    if ('DIOBoard)
        return(-1);
}

```

```

ecb.ECB_StackID = MLID_GET_SN;
ecb.ECB_Fragment[0].FragmentAddress = serialNum;

/* Call MLID */
ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
return(ccode);

```

```

)

```

```

/*****

```

```

* DIOSignText(char *textToSign,
*             LONG textLength,
*             char *signature)

```

```

* Description:

```

```

    This routine uses the adapter to calculate a signature
    for the given text.
    It is used for explicit requests of packages that are
    for sale.

```

```

* Input:

```

```

    textToSign      - string to be signed
    textLength      - length of string to be signed
    signature       - string to store signature

```

```

* Output:

```

```

    signature       - filled out if successful

```

```

* Returns:

```

```

    0 if successful

```

```

*****

```

```

LONG DIOSignText(char *textToSign,
                 LONG textLength,
                 char *signature)

```

```

(

```

```

    LONG ccode;
    LONG fragment[3];
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 139)};

```

```

    if (!DIOBoard)
        return(-1);

```

```

    fragment[0] = (LONG)textToSign;
    fragment[1] = textLength;
    fragment[2] = (LONG)signature;

```

```

    ecb.ECB_StackID = MLID_SIGN_TEXT;
    ecb.ECB_Fragment[0].FragmentAddress = fragment;

```

```

    /* Call MLID */
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);

```

```

)

```

```

/*****

```

```

* EXPORTED FUNCTION

```

```

* DPCGetMLIDStats(struct DriverStatsStructure **stats)

```

```

*

```

```

* Description:

```

```

    This routine fills in a pointer to the MLID stats
    table.

```

```

* Input:

```

```

    stats          - Pointer to where to store poin
    ter to stats table

```

```

* Output:

```

```

    stats filled in if successful

```

```

* Returns:

```

```

    0 if MLID is active

```

```

*****

```

```

int DPCGetMLIDStats(struct DriverStatsStructure **stats)

```

```

(

```

```

    if (!DIOBoard)

```

```

        return(-1);

```

```

    *stats = DIOSStats = (struct DriverStatsStructure *)
    CommandMlid(DIOBoard, 1, DIOControlEntry);

```

```

    return(0);

```

```

)

```

```

int DIOGetMLIDConfig(struct DriverConfigurationStructure **config)

```

```

(

```

```

    if (!DIOBoard)

```

```

        return(-1);

```

```

    *config = (struct DriverConfigurationStructure *)
    CommandMlid(DIOBoard, 0, DIOControlEntry);

```

```

    return(0);

```

```

)

```

```

/*****

```

```

* DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)

```

```

* Description:

```

```

    This routine attempts to open an adapter channel for
    the passed in bypass address, such as 0f 00 00 00 00.

```

```

* Input:

```

```

    address        - Bypass

```

```

    address for open

```

```

    esr

```

```

    - ESR address for this channel

```

```

    channel number is returned

```

```

* Output:

```

```

    channel is filled out if successful

```

```

* Returns:

```

```

    0 if channel was opened

```

```

*****

```

```

/*****

```

```

* LONG DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)

```

```

(

```

```

    ChannelConfig cfg;

```

```

*

```

```

LONG
ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 142));

if (!DIOBoard) {
    if (DIORegisterWithAdapter(DPCName))
        return(-1);
}

/* Initialize channel to 0. MLID will overwrite this */
cfg.CfgChannel = 0;

/* Point ESR to our packet handler */
cfg.CfgESR = esr;

/* Number of addresses to add */
cfg.CfgNumAddresses = 1;

/* Address of 0f 00 00 00 00 */
CMovB(address, cfg.CfgAddress, 6);

ecb.ECB_StackID = MLID_OPEN_CHANNEL;
ecb.ECB_Fragment[0].FragmentAddress = &cfg;

/* Call MLID */
ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
if (ccode == 0)
{
    /* Store channel for close channel, add addr and delete addr */
    *channel = cfg.CfgChannel;
    return(ccode);
}

}

/*****
 *
 * DIOCloseChannel(LONG channel)
 *
 * Description: This routine closes a previously opened channel.
 *
 * Input: channel - channel
 *
 * Output: nothing
 *
 * Returns: 0 if channel was closed
 *
 *****/
LONG
{
    DIOCloseChannel(LONG channel)

    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 157));

    if (!DIOBoard)
        return(-1);

    ecb.ECB_StackID = MLID_CLOSE_CHANNEL;
    ecb.ECB_Fragment[0].FragmentAddress = &channel;
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);

    return(ccode);
}

}

LONG DIOAddGroupAddress(LONG channel, BYTE *address, BYTE *groupAddress)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 209));
    BYTE key[8];

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 8);

    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

}

LONG DIOAddAddress(LONG channel, BYTE *address)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 207));

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 8);

    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

}

LONG DIODeleteAddress(LONG channel, BYTE *address)
{
    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 180));

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 6);

    ecb.ECB_StackID = MLID_DELETE_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

}

LONG
{
    DIODeleteAddress(LONG channel, BYTE *address)

    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 180));

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 6);

    ecb.ECB_StackID = MLID_DELETE_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

}

/*****
 *
 * DIODeleteAddress(LONG channel, BYTE *address)
 *
 * Description: This routine deletes the filter address from the MLID.
 *
 * Input: address - address to delete
 *
 * Output: Nothing
 *
 * Returns: 0 if successful
 *
 *****/
LONG
{
    DIODeleteAddress(LONG channel, BYTE *address)

    ChannelConfig channelCfg;
    LONG ccode;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 180));

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 6);

    ecb.ECB_StackID = MLID_DELETE_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

}

```

```

        if (!DIOBoard)
            return(-1);

        channelCfg.CfgChannel = channel;
        channelCfg.CfgNumAddresses = 1;
        CMovB(address, channelCfg.CfgAddress, 8);
        CMovB(groupAddress, key, 8);
        reverse_key((BYTE*)&key);
        CMovB(key, channelCfg.CfgGroupKey, 8);
        CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

        ecb.ECB_StackID = MLID_ADD_ADDRESS;
        ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
        ccode = IoctlMId(DIOBoard, &ecb, DIOControlEntry);
        return(ccode);
    }

    int DIOAddHIAddr(unsigned char channel, BYTE *hiAddr)
    {
        chunk key;
        ID hi_id;
        int i, ret = CAS_ERROR;
        ChannelConfig channelCfg;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 471)};
        if (!DIOBoard)
            return(-1);

        for(i = 0; i < 3; i++)
            hi_id.i[i] = 0x00;
        make_hi_key(&key);
        channelCfg.CfgChannel = channel;
        channelCfg.CfgNumAddresses = 1;

        CMovB(hiAddr, channelCfg.CfgAddress, 8);
        /* Some strange things ... */
        reverse_key((BYTE *)&key);
        CMovB(&key, channelCfg.CfgGroupKey, 8);
        CMovB(&key, channelCfg.CfgElementKey, 8);

        // channelCfg.CfgChannel = FDBChannel;
        // channelCfg.CfgESR = FDB_ESR;
        // channelCfg.CfgNumAddresses = 1;
        // CMovB(&addr, channelCfg.CfgAddress, 8);
        // CMovB(&pacau->g_key, key, 8);
        // reverse_key(&key);
        // CMovB(key, channelCfg.CfgGroupKey, 8);
        // CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

        channelCfg.CfgESR = (int (*)(!))0xffffffff;
        ecb.ECB_StackID = MLID_ADD_ADDRESS;
        ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
        CMovB(&addr, node->file_address, 6);

        ret = IoctlMId(DIOBoard, &ecb, DIOControlEntry);

        // if((ret = WBicddAddAddress
        //      ((BICDD_CHANNEL_CONFIG FAR *)&channel_config))!=CAS_OK)
        //
        // return ret;
    }

    int DIORegisterSend(int (*sendRoutine)(TCB *))
    {
        if (!DIOControlEntry)
            return(-1);

        ecb.ECB_StackID = MLID_REGISTER_SEND_ROUTINE;
        ecb.ECB_Fragment[0].FragmentAddress = &sendRoutine;

        /* Call MLID */
        IoctlMId(DIOBoard, &ecb, DIOControlEntry);
        return(0);
    }

    int DIOObtainReturnTCB(void (**returnTCBRoutine)(TCB *))
    {
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 147)};

        if (!DIOControlEntry)
            return(-1);

        ecb.ECB_StackID = MLID_RETURN_TCB_ROUTINE;
        ecb.ECB_Fragment[0].FragmentAddress = returnTCBRoutine;

        /* Call MLID */
        IoctlMId(DIOBoard, &ecb, DIOControlEntry);
        return(0);
    }

    int DPCGetIPAddress(LONG* ip) {
        LONG id;
        LONG (*ctl)(LONG board, ...);
        char buf[80];
        char* s = buf;

        if (CLSLGetStackIDFromName("\002IP", &id))
            return 1;
        if (CLSLGetProtocolControlEntry(id, DIOBoard, &ctl))
            return 2;
        if (GetProtocolStringForBoard(ctl, DIOBoard, buf))
            return 3;
        s = strbrk(buf, "0123456789");
        if (!s)
            return 4;
        *ip = inet_addr(s);
        if (*ip == (LONG)(-1))
            return 5;
        return 0;
    }
}

```

```
#include "dpcagent.h"
#include <string.h>
#include <stdlib.h>
```

```
LONG DPCMaxConnections = 3;
int PackageDelivery = FALSE;
```

```
DloCfg_t DloCfg = {
    1330, // freq
    (198 << 24) | (77 << 16) | (117 << 8) | 21, // ip_address
    (198 << 24) | (77 << 16) | (117 << 8) | 66, // gateway_address
    1500, // mtu
    MSG("ATDT1-800-332-8071", 100), // tinet_phone_num
    MSG("ATDT1-800-825-3954", 187), // pdeliv_phone_num
    "", // dialout_prefix
    300, // tinet_inactivity_timer
    2, // pdeliv_inactivity_timer
    1, // modem_type
    120, // packet_lifetime - Max time to keep data in buffer
    60, // call_setup_timeout
    MSG("ATH0", 189), // hangup_str
    MSG("NO CARRIER", 190), // disconnect_str
    MSG("+++", 191), // escape_str
    MSG("CONNECT", 192), // connect_str
    MSG("ATE1Q0V1X4&c1&d2 S7=60 S11=55", 193), // init_str
    1024, // max_db_entries
    5, // tinet_baud_index(19200)
    0, // pdeliv_baud_index(2400)
    1024 * 8, // async_buffer_size(8K)
    FALSE, // auto_login
    "", // wait_for_1..wait_for_9
    "", // send_1..send_9
    OUT_SLIP, // out_protocol
    TRUE, // obsolete(was tunnel)
    10, 5, 5, 5, 5, 5, 5, // wait_timeout_1..wait_timeout_9
    "", // ppp_login
    1400, // ppp_mru
    0, // ppp_passwd
    "", // ppp_accm
    "00000000", // base_license
    "", // key
    -1, // net_interface
    "\0\0\0\0\0", // net_addr
    { "", "", "", "", "", "", "", "" }, // add_license
};
```

```
/* ***** EXPORTED FUNCTION ***** */
```

```
DPCUpdateConfig(void)
```

```
Description:
```

```
This routine opens \DIRECTPC\MODEM.CFG and updates our global
structures. If the file doesn't exist, or if its an older versio
n (because its smaller), read in what you can and write out a new
one.
```

```
Input:
```

```
Output:
```

```
* Returns:
* 0 if successful
* -1 unable to open or create file
* -2 unable to update file
```

```
*****
```

```
void ConfigSanityCheck(int handle)
```

```
{
    int changeFlag = 0;
    int i;
    LONG *timeout;

    if (DloCfg.wait_timeout_1 < 2 || DloCfg.wait_timeout_1 > 60)
    {
        changeFlag++;
        DloCfg.wait_timeout_1 = 10;
    }

    for (i = 0, timeout = &DloCfg.wait_timeout_2; i < 8; i++, timeout++)
    {
        if (*timeout < 2 || *timeout > 60)
        {
            changeFlag++;
            *timeout = 5;
        }

        if (changeFlag)
        {
            lseek(handle, 0, SEEK_SET);
            write(handle, &DloCfg, sizeof(DloCfg));
        }
    }

    int DPCUpdateConfig(void)
    {
        int ccode = -1;
        int handle;

        handle = open(MSG("SYS:DIRECPC\DB\MODEM.CFG", 194),
                     O_RDWR | O_CREAT,
                     S_IWWRITE | S_IREAD);
        if (handle != -1)
        {
            if ( read(handle, &DloCfg, sizeof(DloCfg)) != sizeof(DloCfg))
            {
                lseek(handle, 0, SEEK_SET);
                if (write(handle, &DloCfg, sizeof(DloCfg)) != sizeof(DloCfg))
                {
                    ccode = -2;
                }
            }

            ConfigSanityCheck(handle);
            close(handle);
            ccode = 0;
        }

        DloCfg.ip_address = htonl(DloCfg.ip_address);
        DloCfg.gateway_address = htonl(DloCfg.gateway_address);

        return(ccode);
    }

    void DPCUpdateConfigFile(void) {
```

```

int handle;

DloCfg.ip_address = ntohl(DloCfg.ip_address);
DloCfg.gateway_address = ntohl(DloCfg.gateway_address);
handle = open(MSG("SYS:DIRECTPC\VD\MODEM.CFG", 335),
              O_RDWR | O_CREAT,
              S_IWRITE | S_IREAD);

if (handle != -1)
{
    write(handle, &DloCfg, sizeof(DloCfg));
    close(handle);
}

DloCfg.ip_address = htonl(DloCfg.ip_address);
DloCfg.gateway_address = htonl(DloCfg.gateway_address);
}

#define SerialWarn(s) sprintf(warnbuf, "\r\nDPCN: detected a bad serial number: %8.8s\r\n", (char*)(s)), ConsolePrintf(warnbuf), RingTheBell()

static inline unsigned long find_and_clear_low_bit(unsigned long* val) {
    unsigned long low = *val;
    if (low == 0)
        return 0;
    *val &= low - 1;
    return (*val ^ low);
}

static inline int parity(LONG serial) {
    int i;
    for (i = 0; find_and_clear_low_bit(&serial); ++i)
        return (i & 1);
}

static inline int UserCount(BYTE* s) {
    LONG serial = 0;
    int shift;

    s += 3;
    for (shift = 16; shift >= 0; shift -= 4) {
        serial |= (*s - ((*s >= 'A') ? 56 : 0x30)) << shift;
        ++s;
    }
    return 5 * (((serial & 0x000002) >> 1) |
                ((serial & 0x200000) >> 16) |
                ((serial & 0x020000) >> 11) |
                ((serial & 0x000400) >> 3));
}

void DPCSetMaxConnections(LONG* sum) {
    char warnbuf[120];
    int users;
    int pd = 0;
    int i;

    sum[0] = sum[1] = (-1);

    /* re-read modem.cfg file */
    i = DloCfg.ip_address;
    DPCUpdateConfig();
    DloCfg.ip_address = i;

```

```

if (strcmp(DloCfg.base_license, "Helius, Inc.", 8) == ESUCCESS) {
    DPCMaxConnections = 108;
    PackageDelivery = 1;
    memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
    return;
}

/* check for old license info */
if (atoi(DloCfg.base_license) < 02) {
    sprintf(warnbuf,
            "\r\nDPCN: deactivated old serial number: %8.8s\r\n",
            DloCfg.base_license);
    ConsolePrintf(warnbuf);
    RingTheBell();
    return;
}

/* handle base license first */
memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
if (parity(sum[0]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
}
if (parity(sum[1]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
}
users = UserCount(DloCfg.base_license);
if (DloCfg.base_license[2] == '*')
    pd = 1;

/* now handle additive licenses */
for (i = 0; i < 9; ++i) {
    int j;
    if (DloCfg.add_license[i][0] == 0)
        continue;
    /* check for duplicate license */
    for (j = i - 1; j >= 0; --j) {
        if (memcmp(DloCfg.add_license[i],
                  DloCfg.add_license[j],
                  sizeof(DloCfg.add_license[i])) == ESUCCESS) {
            memset(DloCfg.add_license[i], 0, sizeof(DloCfg.add_license[i]));
            DPCUpdateConfigFile();
            sprintf(warnbuf,
                    "\r\nDPCN: deleted duplicate license number: %8.8s\r\n",
                    DloCfg.add_license[j]);
            ConsolePrintf(warnbuf);
            RingTheBell();
            goto nextLicense;
        }
    }
    memcpy(serial, DloCfg.add_license[i], sizeof(serial));
    if (parity(serial[0]) == 1) {
        SerialWarn(DloCfg.add_license[i]);
        return;
    }
    if (parity(serial[1]) == 1) {
        SerialWarn(DloCfg.add_license[i]);
        return;
    }
    users += UserCount(DloCfg.add_license[i]);
    sum[0] += serial[0];
    sum[1] += serial[1];
    if (DloCfg.add_license[i][2] == '*')

```

Thu Jul 17 14:46:12 1997

license.c

Page 5 of 23

```
pd = 1;  
nextLicense:  
    ;  
    )
```

```
DPCMaxConnections = users * 4;  
PackageDelivery = pd;  
    )
```

```

#include <dpccagent.h>
/* Our header file */

#define USE_AIO_DEADMAN 0
#define TRACE_STATE 0

static char* printables = MSG("A..Za..z0..9 -.~!@#$%^&*()_+=[]{}|:'.<?>\\/\\"
523);
#define dial_chars printables
#define modem_control_chars printables

int AIOPortHandle = -1;
LONG AIOWriteBufferSize = 0;
int AIOGlobalPort = 0;
static BYTE AIOBaudRateDefines[] =
{
    AIO_BAUD_2400, /* 0 */
    AIO_BAUD_3600, /* 1 */
    AIO_BAUD_4800, /* 2 */
    AIO_BAUD_7200, /* 3 */
    AIO_BAUD_9600, /* 4 */
    AIO_BAUD_19200, /* 5 */
    AIO_BAUD_38400, /* 6 */
    AIO_BAUD_57600, /* 7 */
    AIO_BAUD_115200 /* 8 */
};

int DloState = DLOS_IDLE;
LONG Dlotimer = 0;
LONG DloPacketLifetime = 0;

LONG DloConn = DLO_CONN_IDLE;
LONG DloNextConn = DLO_CONN_IDLE;

LONG DloPxmCount = 0;
LONG DloMaxBufferSize = DLOBUFSIZE;
LONG DloPxmBuffer[DLOBUFSIZE];
BYTE DloPxmBuffer[DLOBUFSIZE];
LONG DloInactivityTimer = 10 * 19;

LONG DloXmitCount = 0;
LONG DloMaxBufferSize = DLOBUFSIZE;
LONG DloXmitBuffer[DLOBUFSIZE];
LONG DloInactivityTimer = 60 * 19;

LONG DloLastKnownTickCount;
LONG DloRcvCount = 0;
LONG DloRcvIndex = 0;
LONG DloReadIndex = 0;
LONG DloRcvBuffer[DLOBUFSIZE];
LONG DloCommandIndex = 0;
LONG DloCommandBuffer[DLOCMDBUFSIZE];
LONG DloLastDCD = 0;

char ConnectBaudStr[DLOCMDBUFSIZE] = {0};

static char *DloCompareString[DLOENUM] =
{
    "",
    DloCfg.connect_str,
    DloCfg.disconnect_str,
    "",
    MSG("OK", 219),
    MSG("BUSY", 220),
    MSG("NO DIAL TONE", 221),
    MSG("RING", 222),
};

```

```

MSG("NO ANSWER", 99)
);
/*
 * Variables used by DloScheduleReceive().
 */
void (*DloCallBack)(BYTE *pdata, int len, int timeoutFlag) = 0;
int DloCallBackTimeout = 0;
int DloCallBackWait = 0;
BYTE DloCallBackBuffer[4000];
int DloCallBackIndex = 0;
int DloCallBackStarted = 0;
int DloCallBackEscape = 0;
int DloCallBackType = 0;

/*
 * RS232.NLM Default Init String
 */
//BYTE
ModemInitString[] = MSG("ATH0QOV1X4S0", 72);

/*
 * Digitan DS144FVM, Multitech Auto Reliable and Practical
 * Peripherals V.34 Default Init String
 */
//BYTE
ModemInitString[] = "ATE1QOV1X4&C1&D2&K0 S7=60 S
11=55";

/*
 * Hayes-Compatible Modem Default Init String
 */
//BYTE
ModemInitString[] = MSG("ATE1QOV1X4&C1&D2 S7=60
S11=55", 156);

/*
 * Intel 144e Faxmodem and Motorola UDS V.3225
 * Default Init String
 */
//BYTE
ModemInitString[] = "ATE1QOV1X4&C1&D2\G S7=60 S1
1=55";

/*
 * U.S. Robotics Default Init String
 */
//BYTE
ModemInitString[] = MSG("ATE1QOV1X4&C1&D2&K0&H0&
10 S7=60 S11=55", 100);

/*
 * Internal function prototypes */
void InitializeAIO(void);
void SendAIOData(BYTE *data, LONG length);
static void DloStartTimer( LONG ticks );
//void DloStopTimer( void );
static void DloStopPacketLifetime( void );
static void DloStartPacketLifetime( void );
static void StateMachine( int event);
static void WriteCommXmitBuffer( void );

void HexAsciiDump(const unsigned char* buffer, unsigned int len)
{
    char display[80];
    while (len >= 16)
    {
        NWprintf(MSG("%02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x", 127),
            buffer[0],
            buffer[1],

```



```

DloRevCount = 0;
DloRevIndex = 0;
DloReadIndex = 0;
}

void DloStartConn(int timeout)
{
    if (DloNextConn == DLO_CONN_IDLE)
    {
        /* Assume package delivery connection first */
        DloNextConn = DLO_CONN_PACKAGE;
        if (timeout == DLO_PACKAGE_TIMEOUT)
        {
            DloInactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
9;
        }
        else if (timeout == DLO_INET_TIMEOUT)
        {
            DloInactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
            DloNextConn = DLO_CONN_INET;
        }
        else if (timeout == DLO_GETKEYS_TIMEOUT)
        {
            DloInactivityTimer = 30 * 19;
        }
        else if (timeout > 2)
        {
            DloInactivityTimer = timeout * 19;
        }
        else
        {
            DloInactivityTimer = 2 * 19;
        }
    }

    if (DloState == DLOS_CONN && DloNextConn == DLO_CONN)
    {
        DloNextConn = DLO_CONN_IDLE;
        StateMachine(DLOE_SEND);
        return;
    }

    if (DloConn == DLO_CONN_IDLE)
    {
        StateMachine(DLOE_SEND);
        DloConn = DloNextConn;
        DloNextConn = DLO_CONN_IDLE;
    }
    else if (DloConn == DLO_CONN_INET)
    {
        /* Package waiting for internet. Cause it to timeout quickly */
        DloStartTimer(19);
    }

    BaudRateHandler(int option, void *parameter)
    {
        parameter = parameter;
        return option;
    }

    static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
    {

```

```

        int i;
        void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
        int save;
        DloCfg_t tmpDloCfg;
        MFCONTROL *mfctl1;
        int baud;

        CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

        NWSInitForm(NUTHandle);

        NWSGetListSortFunction(NUTHandle, &oldSortFunction);
        NWSSetListSortFunction(NUTHandle, NoSortHandler);

        i = 0;
        NWSAppendCommentField(i, 2, MSG("Package Phone
andle);
        NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg.pdeliv_phone_num
,
            dial_chars, F_NO_HELP, NUTHandle);

        i++;
        baud = tmpDloCfg.pdeliv_baud_index;
        NWSAppendCommentField(i, 2, MSG("Package Baud
andle);
        mfctl1 = NWSInitMenuField(InxMSG("Baud Rate", 314), 10, 40, BaudRateHand
ler, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-2400", 315), 0, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-3600", 316), 1, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-4800", 317), 2, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-7200", 318), 3, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-9600", 319), 4, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-19200", 320), 5, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-38400", 321), 6, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-57600", 322), 7, NUTHandle);
        NWSAppendToMenuField(mfctl1, InxMSG("-115200", 323), 8, NUTHandle);
        NWSAppendMenuField(i, 28, NORMAL_FIELD, &baud, mfctl1, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Package Inactivity(sec) : ", 324), NUTH
andle);
        NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.pdeliv_inac
tivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

        i++;
        NWSAppendCommentField(i, 2, MSG("Max Database Entries : ", 327), NUTH
andle);
        NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.max_db_entr
ies, 1, 65535, F_NO_HELP, NUTHandle);

        i++;
        save = NWSeditPortalForm(InxMSG("Package Delivery Configuration Editor",
308),
            12, 40,
            /* center line, column */
            i, 76,

```



```

    LONG net_addr[6];
    void (*ControlEntryPoint)(void) = 0;
    struct DriverConfigurationStructure* dvrCfg = 0;
    if (sscanf(hw_addr, "%2x-%2x-%2x-%2x-%2x-%2x",
        &net_addr[0],
        &net_addr[1],
        &net_addr[2],
        &net_addr[3],
        &net_addr[4],
        &net_addr[5]) != 6)
    {
        NWSAlert(12, 40, NUTHandle,
            InxMSG("Format error in Mac Address", 680));
        goto retry;
    }
    for (i = 0; i < 6; ++i)
        tmpDlocfg->net_addr[i] = (BYTE)net_addr[i];
    if (CPLGetMLIDControlEntry(interface,
        &ControlEntryPoint))
    {
        NWSAlert(12, 40, NUTHandle,
            InxMSG("Interface not found", 681));
        goto retry;
    }
    dvrCfg = (struct DriverConfigurationStructure *)
        CommandMlId(interface, 0, (LONG)ControlEntryPoint);
    if (!dvrCfg)
    {
        NWSAlert(12, 40, NUTHandle,
            InxMSG("Could not retrieve Interface Configurati
ion Table", 682));
        goto retry;
    }
    if ((dvrCfg->DModeFlags & (1 << 6)) == 0)
    {
        NWSAlert(12, 40, NUTHandle,
            InxMSG("Interface does not support 'Raw Send\"
", 683));
        goto retry;
    }
    if (dvrCfg->DMediaID != 2)
    {
        NWSAlert(12, 40, NUTHandle,
            InxMSG("Interface does not support ETHERNET_II
frames", 684));
        goto retry;
    }
    if (!ICStrCmp(dvrCfg->DShortName, DPCName))
    {
        NWSAlert(12, 40, NUTHandle,
            InxMSG("Do not use DPC as the Interface", 685))
        goto retry;
    }
    tmpDlocfg->net_interface = interface;
}

NWSDestroyForm(NUTHandle);
NWSPopList(NUTHandle);
return K_SELECT;
}

LONG
ModifyLoginScript(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
    int i;
    LONG save;
    *tmpDlocfg;

    dlocfg_t
    LONG save;

    key = key;
    changed = changed;
    handle = handle;

    tmpDlocfg = (Dlocfg_t *)fp->customData;
    if (NWSPushList(NUTHandle) == 0)
        return K_SELECT;

    NWSInitForm(NUTHandle);

    i = 0;

    NWSAppendCommentField(i, 2, MSG("Wait1: ", 581), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_1,
        printables, F_NO_HELP, NUTHandle);

    NWSAppendCommentField(i, 40, MSG("Wait Timeout 1: ", 599), NUTHandle);
    NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_1, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Send1: ", 518), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->send_1,
        printables, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Wait2: ", 520), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_2,
        printables, F_NO_HELP, NUTHandle);
    NWSAppendCommentField(i, 40, MSG("Wait Timeout 2: ", 600), NUTHandle);
    NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_2, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Send2: ", 522), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->send_2,
        printables, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Wait3: ", 524), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_3,
        printables, F_NO_HELP, NUTHandle);
    NWSAppendCommentField(i, 40, MSG("Wait Timeout 3: ", 601), NUTHandle);
    NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_3, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Send3: ", 526), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->send_3,
        printables, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Wait4: ", 528), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->wait_for_4,
        printables, F_NO_HELP, NUTHandle);
    NWSAppendCommentField(i, 40, MSG("Wait Timeout 4: ", 602), NUTHandle);
    NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDlocfg->wait_timeo
ut_4, 2, 60, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Send4: ", 530), NUTHandle);
    NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDlocfg->send_4,

```

```
printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait5: ", 532), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_5,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 5: ", 603), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_5, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send5: ", 534), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_5,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait6: ", 536), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_6,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 6: ", 604), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_6, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send6: ", 538), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_6,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait7: ", 540), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_7,
    printables, F_NO_HELP, NUTHandle);
```

```
NWSAppendCommentField(i, 40, MSG("Wait Timeout 7: ", 605), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_7, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send7: ", 542), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_7,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait8: ", 544), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_8,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 8: ", 606), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_8, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send8: ", 546), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_8,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait9: ", 548), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_9,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 9: ", 607), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_9, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send9: ", 550), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_9,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
save = NWSeditPortalForm(InxMSG("Login Script Editor", 582),
    12, 40,
    /* center line, column */
    /* form height, width */
    F_NOVERIFY, F_NO_HELP, /* Control flags, help message */
    InxMSG("Save Changes?", 583),
    NUTHandle); /* Confirm message, hand
```

```
le */
// if (save)
// {
//     CMovB(tmpDloCfg->wait_for_1, DloCfg.wait_for_1, 30*18);
//     CMovB(&tmpDloCfg->wait_timeout_1, &DloCfg.wait_timeout_1, 9 * si
//         zeof(LONG));
//     DPCUpdateConfigFile();
//     NWSDestroyForm(NUTHandle);
//     NWSPopList(NUTHandle);
//     return K_SELECT;
// }
int NewProtocolFlag = -1;
```

```
LONG ChangeProtocol(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
```

```
    DloCfg_t *tmpDloCfg;
    LIST *listPtr, *slipList, *pppList, *netList;
    LONG rcode;
    LONG rcode = K_SELECT;
```

```
    key = key;
    changed = changed;
    handle = handle;
```

```
    tmpDloCfg = (DloCfg_t *)fp->customData;
```

```
    if (NWSPushList(NUTHandle) == 0)
        return rcode;
```

```
    NWSInitList(NUTHandle, NULL);
```

```
    slipList = NWSAppendToList(MSG("Modem - SLIP", 519), (void *)OUT_SLIP, N
        UTHandle);
```

```
    pppList = NWSAppendToList(MSG("Modem - PPP", 521), (void *)OUT_PPP, NUTH
        andle);
    netList = NWSAppendToList(MSG("LAN/WAN", 537), (void *)OUT_NETWORK, NUTH
        andle);
```

```
    if (tmpDloCfg->out_protocol == OUT_SLIP)
```

```
    {
        listPtr = slipList;
```

```
    }
    else if (tmpDloCfg->out_protocol == OUT_PPP)
```

```
    {
        listPtr = pppList;
```

```
    }
    else
```

```
    {
        listPtr = netList;
```

```
    }
```

```
    ccode = NWSList(
```

```

InxMSG("Outbound Protocol", 509),
12, 40,
3,
16,
M_ESCAPE | M_SELECT,
&listPtr,
NUTHandle, NULL,
NULL, NULL);

if (ccode == M_SELECT)
{
    tmpDlocfg->out_protocol = NewProtocolFlag = (int)listPtr->otherI
    rcode = K_ESCAPE;
}

NWSDestroyList(NUTHandle);
NWSPopList(NUTHandle);
return rcode;
}

//LONG ChangeTunnel(FIELD *fp, int key, int *changed, NUTInfo *handle)
//{
//    Dlocfg_t *tmpDlocfg;
//    LIST *listPtr, *enableList, *disableList;
//    LONG ccode;
//    LONG rcode = K_SELECT;
//
//    key = key;
//    changed = changed;
//    handle = handle;
//
//    tmpDlocfg = (Dlocfg_t *)fp->customData;
//
//    if (NWSPushList(NUTHandle) == 0)
//        return rcode;
//
//    NWSInitList(NUTHandle, NULL);
//
//    enableList = NWSAppendToList(MSG("Enabled", 624), (void *)1, NUTHandle);
//    disableList = NWSAppendToList(MSG("Disabled", 625), (void *)0, NUTHandle);
//};
//
//    if (tmpDlocfg->tunnel)
//    {
//        listPtr = enableList;
//    }
//    else
//    {
//        listPtr = disableList;
//    }
//
//    ccode = NWSList(
//        InxMSG("Tunnel Header", 626),
//        12, 40,
//        2,
//        16,
//        M_ESCAPE | M_SELECT,
//        &listPtr,
//        NUTHandle, NULL,
//        NULL, NULL);
//
//    if (ccode == M_SELECT)
//    {
//        tmpDlocfg->tunnel = NewProtocolFlag = (int)listPtr->otherInfo;
//        rcode = K_ESCAPE;
//    }
//
//    InxMSG("Outbound Protocol", 509),
//    12, 40,
//    3,
//    16,
//    M_ESCAPE | M_SELECT,
//    &listPtr,
//    NUTHandle, NULL,
//    NULL, NULL);
//
//    if (ccode == M_SELECT)
//    {
//        tmpDlocfg->out_protocol = NewProtocolFlag = (int)listPtr->otherI
//        rcode = K_ESCAPE;
//    }
//
//    NWSDestroyList(NUTHandle);
//    NWSPopList(NUTHandle);
//    return rcode;
//}

void ProviderConfiguration(void)
{
    int i;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
    int save; /*
    Dlocfg_t tmpDlocfg;
    int ip0, ip1, ip2, ip3;
    int gw0, gw1, gw2, gw3;
    MFCNTROL *mfct10;
    int baud;
    FIELD *fp;
    char *protocolStr;
    char *pppConfigStr;
    int cflags = F_VERIFY;
    char *tunnelStr;

    CMovB(&Dlocfg, &tmpDlocfg, sizeof(Dlocfg_t));

    NewProtocolLoop:
    NewProtocolFlag = -1;

    NWSInitForm(NUTHandle);

    NWSGetListSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 30, MSG("Outbound Protocol : ", 525), NUTHandle);

    if (tmpDlocfg.out_protocol == OUT_SLIP)
        protocolStr = MSG("Modem - SLIP", 527);
    else if (tmpDlocfg.out_protocol == OUT_PPP)
        protocolStr = MSG("Modem - PPP", 529);
    else
        protocolStr = MSG("LAN/WAN", 584);

    fp = NWSAppendHotSpotField(i, 50, NORMAL_FIELD, protocolStr,
        ChangeProtocol, NUTHandle);

    fp->customData = &tmpDlocfg;

    i+=2;
    NWSAppendCommentField(i, 2, MSG("Internet Phone
    THandle);
    NWSAppendStringField(i, 30, 30, NORMAL_FIELD, tmpDlocfg.tinet_phone_num,
        dial_chars, F_NO_HELP, NUTHandle);

    i++;
    baud = tmpDlocfg.tinet_baud_index;
    NWSAppendCommentField(i, 2, MSG("Internet Baud
    THandle);
    mfct10 = NWSInitMenuField(InxMSG("Baud Rate", 129), 10, 40, BaudRateHand
    ler, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("2400", 130), 0, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("3600", 131), 1, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("4800", 132), 2, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("7200", 133), 3, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("9600", 146), 4, NUTHandle);

```

```

NWSAppendToMenuField(mfct10, InxMSG("19200", 297), 5, NUTHANDLE);
NWSAppendToMenuField(mfct10, InxMSG("38400", 300), 6, NUTHANDLE);
NWSAppendToMenuField(mfct10, InxMSG("57600", 304), 7, NUTHANDLE);
NWSAppendToMenuField(mfct10, InxMSG("115200", 305), 8, NUTHANDLE);
NWSAppendMenuField(i, 30, NORMAL_FIELD, &baud, mfct10, NULL, NUTHANDLE);

i++;
NWSAppendCommentField(i, 2, MSG("Internet MTU", 309), NU
THANDLE);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg.mtu, 1, 655
35, F_NO_HELP, NUTHANDLE);

i++;
NWSAppendCommentField(i, 2, MSG("Internet Inactivity(sec)", 310), NU
THANDLE);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDloCfg.tinet_inact
ivity_timer, 1, 65535, F_NO_HELP, NUTHANDLE);

i++;
NWSAppendCommentField(i, 2, MSG("IP to IP tunneling", 537), NU
THANDLE);
NWSAppendBoolField(i, 30, NORMAL_FIELD, (BYTE *)&tmpDloCfg.tunnel, F_NO_
HELP, NUTHANDLE);
if (tmpDloCfg.tunnel)
    tunnelStr = MSG("Enabled", 627);
else
    tunnelStr = MSG("Disabled", 628);

fp = NWSAppendHotSpotField(i, 30, NORMAL_FIELD, tunnelStr,
ChangeTunnel, NUTHANDLE);
fp->customData = &tmpDloCfg;
if (tmpDloCfg.tunnel)
{
    LONG ip_address = ntohl(tmpDloCfg.ip_address);
    LONG gateway_address = ntohl(tmpDloCfg.gateway_address);
    i++;
    ip0 = (ip_address) & 0xff;
    ip1 = (ip_address >> 8) & 0xff;
    ip2 = (ip_address >> 16) & 0xff;
    ip3 = (ip_address >> 24) & 0xff;
    NWSAppendCommentField(i, 2, MSG("IP Address(ISP)",
306), NUTHANDLE);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, &ip3, 1, 255, F_NO_HE
LP, NUTHANDLE);
    NWSAppendIntegerField(i, 34, NORMAL_FIELD, &ip2, 1, 255, F_NO_HE
LP, NUTHANDLE);
    NWSAppendIntegerField(i, 38, NORMAL_FIELD, &ip1, 1, 255, F_NO_HE
LP, NUTHANDLE);
    NWSAppendIntegerField(i, 42, NORMAL_FIELD, &ip0, 1, 255, F_NO_HE
LP, NUTHANDLE);

    i++;
    gw0 = (gateway_address) & 0xff;
    gw1 = (gateway_address >> 8) & 0xff;
    gw2 = (gateway_address >> 16) & 0xff;
    gw3 = (gateway_address >> 24) & 0xff;
    NWSAppendCommentField(i, 2, MSG("Hybrid Gateway Address",
307), NUTHANDLE);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, &gw3, 1, 255, F_NO_HE
LP, NUTHANDLE);
    NWSAppendIntegerField(i, 34, NORMAL_FIELD, &gw2, 1, 255, F_NO_HE
LP, NUTHANDLE);
    NWSAppendIntegerField(i, 38, NORMAL_FIELD, &gw1, 1, 255, F_NO_HE
LP, NUTHANDLE);
    NWSAppendIntegerField(i, 42, NORMAL_FIELD, &gw0, 1, 255, F_NO_HE
LP, NUTHANDLE);
}
}

i++;
protocolStr = MSG("Modify Auto Login Script", 535);
fp = NWSAppendHotSpotField(i, 25, NORMAL_FIELD, protocolStr,
ModifyLoginScript, NUTHANDLE);
fp->customDataRelease = NWSFree;
fp->customData = &tmpDloCfg;

i++;
if (tmpDloCfg.out_protocol == OUT_PPP)
{
    pppConfigStr = MSG("Modify PPP Configuration", 608);
    fp = NWSAppendHotSpotField(i, 26, NORMAL_FIELD, pppConfigStr,
ModifyPPPConfig, NUTHANDLE);
    fp->customData = &tmpDloCfg;
}
else if (tmpDloCfg.out_protocol == OUT_NETWORK)
{
    fp = NWSAppendHotSpotField(i, 26, NORMAL_FIELD,
"Modify Network Configuration",
ModifyNetConfig, NUTHANDLE);
    fp->customData = &tmpDloCfg;
}

i++;
/* save = */ NWSeditPortalForm(InxMSG("Provider Configuration Editor", 55
5),
12, 40,
/* center line, column */
i, 78,
/* form height, width */
/* cflags*/F_NOVERIFY, F_NO_HELP,
/* Contr
ol flags, help message */
/* InxMSG("Save Changes?", 556)*/ NULL, /* Confirm message, hand
NUTHANDLE);

NWSSetListSortFunction(NUTHANDLE, oldSortFunction);
NWSDestroyForm(NUTHANDLE);
if (NewProtocolFlag != -1)
{
    cflags = F_FORCE;
    goto NewProtocolloop;
}

if (!save)
    return;

tmpDloCfg.ip_address = htonl((ip0 |
(ip1 << 8) |
(ip2 << 16) |

```

```

tmpDlocCfg.gateway_address = htonl((gw0) |
    (gw1 << 8) |
    (gw2 << 16) |
    (gw3 << 24));

tmpDlocCfg.tinet_baud_index = baud;

if (memcmp(&DlocCfg, &tmpDlocCfg, sizeof(DlocCfg)) == 0 ||
    NWSConfirm(InxMSG("Save Changes?", 612), 0, 0, TRUE, NULL,
        NUTHandle, NULL) == FALSE)
    return;

/*
 * Get newest wait_for and send strings in case ModifyLoginScript()
 * changed them.
 */

// CMovB(DlocCfg.wait_for_1, tmpDlocCfg.wait_for_1, 30 * 18);
// CMovB(&DlocCfg.wait_timeout_1, &tmpDlocCfg.wait_timeout_1, 9 * sizeof(LONG
));

CMovB(&tmpDlocCfg, &DlocCfg, sizeof(DlocCfg_t));
InetChangeProtocol();
DPCUpdateConfigFile();

void ModemConfiguration(void)
{
    int i;
    int save;
    tmpDlocCfg;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);

    CMovB(&DlocCfg, &tmpDlocCfg, sizeof(DlocCfg_t));
    NWSInitForm(NUTHandle);

    NWSGetListSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 2, MSG("Packet Life(sec) : ", 325), NUTHandle);

    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlocCfg.packet_life
        time, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Call Setup(sec) : ", 326), NUTHandle);

    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlocCfg.call_setup_
        timeout, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Aync Buffer Size : ", 212), NUTHandle);

    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDlocCfg.async_buffe
        r_size, 1500, 1024*100, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Dial Prefix : ", 329), NUTHandle);

    NWSAppendStringField(i, 24, 10, NORMAL_FIELD, tmpDlocCfg.dialout_prefix,

```

```

modem_control_chars, F_NO_HELP, NUTHandle);

```

```

e);
NWSAppendCommentField(i, 2, MSG("Hangup Str : ", 330), NUTHandle);
NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlocCfg.hangup_str,
    modem_control_chars, F_NO_HELP, NUTHandle);

```

```

e);
NWSAppendCommentField(i, 2, MSG("Disconnect str : ", 331), NUTHandle);
NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlocCfg.disconnect_str,
    modem_control_chars, F_NO_HELP, NUTHandle);

```

```

e);
NWSAppendCommentField(i, 2, MSG("Escape Str : ", 332), NUTHandle);
NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDlocCfg.escape_str,
    modem_control_chars, F_NO_HELP, NUTHandle);

```

```

e);
NWSAppendCommentField(i, 2, MSG("Connect str : ", 333), NUTHandle);
NWSAppendStringField(i, 24, 50, NORMAL_FIELD, tmpDlocCfg.connect_str,
    modem_control_chars, F_NO_HELP, NUTHandle);

```

```

e);
NWSAppendCommentField(i, 2, MSG("Init str : ", 334), NUTHandle);
NWSAppendStringField(i, 24, 60, NORMAL_FIELD, tmpDlocCfg.init_str,
    modem_control_chars, F_NO_HELP, NUTHandle);

```

```

/* center line, column */
save = NWSeditPortalForm(InxMSG("Modem Configuration Editor", 513),
    12, 40,
    i, 76,
    /* form height, width */
    F_VERIFY, F_NO_HELP,
    message */
    InxMSG("Save Changes?", 514),
    NUTHandle);
/* Confirm message, hand

```

```

le */
NWSSetListSortFunction(NUTHandle, oldSortFunction);
NWSDestroyForm(NUTHandle);
if (!save)
    return;
CMovB(&tmpDlocCfg, &DlocCfg, sizeof(DlocCfg_t));
if (AIOPortHandle != -1)
{
    AIOSetWriteBufferSize(AIOPortHandle, DlocCfg.async_buffer_size);
    AIOGetWriteBufferSize(AIOPortHandle, &AIOWriteBufferSize);
    DlocMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
        BufferSize : DLOBUFSIZE;
    DlocMaxBufferSize = (AIOWriteBufferSize < DLOBUFSIZE) ? AIOWrite
        BufferSize : DLOBUFSIZE;
}
DPCUpdateConfigFile();

```


This routine is terminate a modem connection.

```

*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
*
* *****/
void DloEndConn(void)
{
    if (AIOPortHandle < 0)
        return;

    if (DloConn == DLO_CONN_PACKAGE)
        DloPxmmitCount = 0;
    else
        DloIXmitCount = 0;

    switch(DloState)
    {
        case DLOS_INIT:
        case DLOS_REDL:
        case DLOS_DIAL:
        case DLOS_CONN:
            AIOFlushBuffers(AIOPortHandle, (AIO_FLUSH_WRITE_BUFFER |
            AIO_FLUSH_READ_BUFFER));
            DloState = DLOS_CONN;
            StateMachine(DLOE_TIMEOUT);
            break;
        case DLOS_IDLE:
        case DLOS_DISC_1:
        case DLOS_DISC_2:
        case DLOS_DISC_3:
        case DLOS_DISC_4:
            StateMachine(DLOE_DISCONN);
            break;
    }
}

/*****
*
* _0003(void)      In IDLE state, got SND event
*
* Description:
* The state machine is in the IDLE state.
* We've received a SND request.
*
* Input:      nothing
*
* Output:     nothing
*
* Returns:    nothing
*
* *****/
int DloConnected(void)
{

```

LONG extStatus = 0, chgdExtStatus;

```

    if (AIOPortHandle < 0)
        return(FALSE);

    AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus);
    if (extStatus & AIO_EXTSTA_DCD)
        return(TRUE);

    return(FALSE);
}

static void _0003 (void)
{
    LONG extStatus, chgdExtStatus;

    InitializeAIO();
    if (AIOPortHandle < 0)
    {
        UpdateModemStr(MSG("Modem Status: ERROR - Unable to initialize A
        \n", 238));
        return;
    }

    if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
    {
        AIOFlushBuffers(AIOPortHandle,
            (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER)
        );

        SendAIOData(DloCfg.init_str, CStrLen(DloCfg.init_str));
        SendAIOData(MSG("\r", 614), 1);
        DloStartTimer(5 * 19);
        DloState = DLOS_INIT;
        if (DloConn == DLO_CONN_INET)
            InetStateChange(DLOS_INIT);
        UpdateModemStr(MSG("Modem Status: Initializing Modem
        \n", 561));
    }
    else
    {
        if (DloConn == DLO_CONN_PACKAGE)
            DloStartTimer(DloInactivityTimer);
        else
            DloStartTimer(DloInactivityTimer);
        DloStartTimer(30*19);
        DloStopPacketLifeTimer();

        WriteCommXmitBuffer();
        DloState = DLOS_CONN;
        if (DloConn == DLO_CONN_INET)
            InetStateChange(DLOS_CONN);
        if (DloConn == DLO_CONN_PACKAGE)
            UpdateModemStr(MSG("Modem Status: Connected to Package D
            \n", 562));
        else
            UpdateModemStr(MSG("Modem Status: Connected to Internet
            \n", 473));
    }
    {
        if (ConnectBaudStr[0])
        {
            BYTE connectStr[80];

            NWSprintf(connectStr, MSG("Modem Status: Connect
            ed to Internet at%.24s\n", 563), ConnectBaudStr);

```

```

    )
    else
    {
        UpdateModemStr(MSG("Modem Status: Connected to I
        \n", 564));
    }
}

/*
 * _0100(void)          In INIT state, got TIMEOUT
 *
 * Description:
 * The state machine is in the INIT state.
 * We timed out waiting for the modem init sequence.
 *
 * Input:      nothing
 * Output:     nothing
 * Returns:    nothing
 */
static void _0100(void)
{
    LONG extStatus, chgdExtStatus;

    if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
        || !(extStatus & AIO_EXTSTA_DCD))
    {
        WriteCommPhoneNumber();
        DloStartTimer(DloCfg.call_setup_timeout * 19);
        DloState = DLOS_DIAL;
        if (DloConn == DLO_CONN_INET)
            InetStateChange(DLOS_DIAL);
    }
    else
    {
        UpdateModemStr(MSG("Modem Status: Error - Still Connected
        \n", 242));
        DloEndConn();
    }
}

/*
 * _0104(void)          In INIT state, got OK response from mode
 *
 * Description:
 * The state machine is in the INIT state.
 * We've received an OK response from sending modem init sequence.
 *
 * Input:      nothing
 * Output:     nothing
 */

```

```

    Returns:    nothing
 */
static void _0104(void)
{
    WriteCommPhoneNumber();

    DloStartTimer(DloCfg.call_setup_timeout * 19);
    DloState = DLOS_DIAL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DIAL);
}

/*
 * _0200(void)          In DIAL state, got TIMEOUT
 *
 * Description:
 * The state machine is in the DIAL state.
 * It timed out waiting for connect.
 *
 * Input:      nothing
 * Output:     nothing
 * Returns:    nothing
 */
static void _0200(void)
{
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
    SendAIOData(MSG("\r", 615), 1);
    DloStartTimer(10 * 18);
    DloState = DLOS_REDL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_REDL);
}

/*
 * _0201(void)          In DIAL state, got CONNECT response from
 * modem
 *
 * Description:
 * The state machine is in the DIAL state.
 * We've received a CONNECT response from sending ATDT sequence.
 *
 * Input:      nothing
 * Output:     nothing
 * Returns:    nothing
 */
static void _0201(void)

```



```
{
    static void _0207(void)
    {
        UpdateModemStr(MSG("Modem Status: Ringing!!!
\n", 247));
    }
}

/*****
 *
 * _0208(void)
 *
 * In DIAL state, got NO ANSWER response from modem
 *
 * Description:
 * The state machine is in the DIAL state.
 * We've received a NO ANSWER response from sending ATDT sequence.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0208(void)
{
    UpdateModemStr(MSG("Modem Status: No ANSWER
\n", 248));
}

/*****
 *
 * _0300(void)
 *
 * In REDIAL state, got TIMEOUT
 *
 * Description:
 * The state machine is in the REDIAL state.
 * We timed out.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0300(void)
{
    UpdateModemStr(MSG("Modem Status: Timeout - Reinitializing the modem
\n", 249));

    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
    SendAIOData(DIoCfg.init_str, CStrlLen(DIoCfg.init_str));
    SendAIOData(MSG("\r", 616), 1);
    DloStartTimer(5 * 19);
    DloState = DLOS_INIT;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_INIT);
}
```

```

}

/*****
 *
 * _0302(void)
 *
 * In REDIAL state, got Disconnected
 *
 * Description:
 * The state machine is in the REDIAL state.
 * We got disconnected by the remote site.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0302(void)
{
    DloEndConn();
}

/*****
 *
 * _0104(void)
 *
 * In REDIAL state, got OK response from modem
 *
 * Description:
 * The state machine is in the REDIAL state.
 * We've received an OK response from sending modem init sequence.
 *
 * Input: nothing
 *
 * Output: nothing
 *
 * Returns: nothing
 *
 *****/
static void _0304(void)
{
    WriteCommPhoneNumber();
    DloStartTimer(DIoCfg.call_setup_timeout * 19);
    DloState = DLOS_DIAL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DIAL);
}

/*****
 *
 * _0400(void)
 *
 * In CONNECTED state, timed out
 *
 * Description:
 * State: The state machine is in the CONNECTED state.
 * Event: We timed out due to inactivity.
 * Action: Add a 1.5 sec pre-escape delay. DLOS_DISC_1 state.
 *
 * Input: nothing
 *
 * Output:
 */
```



```

BYTE *xmitBuffer;

if (DloNextConn == DLO_CONN_IDLE)
{
    /* Assume package delivery connection first */
    DloNextConn = DLO_CONN_PACKAGE;

    if (timeout == DLO_PACKAGE_TIMEOUT)
    {
        DloInactivityTimer = Dlocfg.pdeliv_inactivity_timer * 1
    }
    else if (timeout == DLO_INET_TIMEOUT ||
             timeout == DLO_INET_NO_TIMEOUT)
    {
        DloInactivityTimer = Dlocfg.tinet_inactivity_timer * 19
    }
    DloNextConn = DLO_CONN_INET;
    else if (timeout == DLO_GETKEYS_TIMEOUT)
    {
        DloInactivityTimer = 30 * 19;
    }
    else if (timeout > 2)
    {
        DloInactivityTimer = timeout * 19;
    }
    else
    {
        DloInactivityTimer = 2 * 19;
    }
}

/* USE_AIO_DEADMAN
*/
* Update AIO deadman timer to timeout + 5 seconds
*/
AIOSetExternalControl(AIOPortHandle, AIO_SET_DEADMAN_TIMER, timeout + 5)

#endif

if (size > DLOBUFSIZE || size < 1)
    return 0;

if (DloState == DLOS_CONN && DloNextConn == DloConn)
{
    if (timeout != DLO_INET_NO_TIMEOUT)
    {
        DloNextConn = DLO_CONN_IDLE;
        StateMachine(DLOE_SEND);
    }
    UpdateModemLights(1, 0, 1);
    if (DebugFlag)
    {
        printf(MSG("Sending %d bytes:\n", 485), size);
        HexAsciiDump(buffer,
                      (DloConn == DLO_CONN_INET && size > 32) ? 3
                      : size);
    }
    SendAIOData(buffer, size);
    return size;
}

maxBufferSize = &DloMaxBufferSize;
xmitCount = &DloPXmitCount;
xmitBuffer = DloPXmitBuffer;
}
else
{
    maxBufferSize = &DloIMaxBufferSize;
    xmitCount = &DloIXmitCount;
    xmitBuffer = DloIXmitBuffer;
}

if (size >= *maxBufferSize - *xmitCount)
    return 0;
for ( i = 0; (i < size) && (*xmitCount < *maxBufferSize); i++, (*xmitCount
t)++)
{
    if (!*xmitCount)
        DloStartPacketLifeTimer();
    xmitBuffer[*xmitCount] = buffer[i];
}

if (DloConn == DLO_CONN_IDLE)
{
    StateMachine(DLOE_SEND);
    DloConn = DloNextConn;
    DloNextConn = DLO_CONN_IDLE;
}
else if (DloConn == DLO_CONN_INET)
{
    /* Package waiting for internet. Cause it to timeout quickly */
    DloStartTimer(19);
}
return size;
}

/*****
WriteCommXmitBuffer(void)
Description:
    This routine is called after the modem is connected to send the
    data stored in Dlo?XmitBuffer to the modem.
Input:
    nothing
Output:
    nothing
Returns:
    nothing
*****/
static void WriteCommXmitBuffer( void )
{
    BYTE *xmitBuffer;
    LONG *xmitCount;

    if (DloConn == DLO_CONN_PACKAGE)
    {
        xmitBuffer = DloPXmitBuffer;
        xmitCount = &DloPXmitCount;
    }
}

```



```
void SendAIOData(BYTE *data, LONG length)
```

```

LONG    count;
int     bufferSize;
BYTE *ptr;
WORD    state;
LONG    bytesWritten;

ptr = data;
while (ptr < (data + length))
{
    AIOWriteStatus(AIOPortHandle, &count, &state);
    bufferSize = AIOWriteBufferSize - count;
    if (length < bufferSize)
        bufferSize = length;
    if (bufferSpace > 0)

```

```
ten);
    AIOWriteData(AIOPortHandle, ptr, bufferSize, &bytesWritten);
    ptr += bufferSize;
}
else
    ThreadSwitchWithDelay();
```

```
AIOPortInfo(int portChoice,
            int *hardwareType,
            int *boardNumber,
            int *portNumber)
```

Description:	The routine returns the AIO information of the port passed in portChoice.
--------------	---

```

Input:
portChoice      - port to return data ab
hardwareType    - where to return hardware type
boardNumber     - where to return board
portNumber      - where to return port n

```

```
Output: hardwareType, boardNumber and portNumber filled in if successful

Returns: 0 if successful
```

```
int AIOPortInfo(int portChoice,
                int *hardwareType,
                int *boardNumber,
                int *portNumber)
```

```
int ccode;
AIOPORTINFO portInfo;
AIOPORTSEARCH portSearch;
```

```
portInfo.returnLength = sizeof (AIOPORTINFO);
```

```
ccode = AIOGetFirstPortInfo(-1, -1, -1, &portSearch, &portInfo,  
                             NULL, NULL, portOwner);
```

```
if (ccode)
    return (-1);
```

```
while (!code)
```

```

if (portInfo.portNumber == portChoice)
{
    if (portInfo.availability == AIO_AVAILABLE_FOR_ACQUIRE)
    {
        *hardwareType = portInfo.hardwareType;
        *boardNumber = portInfo.boardNumber;
        *portNumber = portInfo.portNumber;
        return 0;
    }
}

```

```
return (-2);
```

```

    }
    ccode = AIOGetNextPortInfo(&portSearch, &portInfo, NULL, NULL,
                                portOwner);
}
return -1;

```

```
InitializeAIO(void)
```

Field	Description
Initialize AIO	If it didn't initialize, AIOPortHandle will still be negative.

```
Input:  nothing
Output: nothing
Returns: nothing
```

```
void InitializeAIO(void)
```

```
int ccode;
int hardware = AIO_HARDWARE_TYPE_WILDCARD;
int port = AIO_PORT_NUMBER_WILDCARD;
int board;
AIO_DRIVERLIST dvr;
dvr.returnLength = sizeof(AIO_DRIVERLIST);

if (AIOPortHandle >= 0)
    return;
```

```
while (AIOGetDriverList(hardware, &dvr) == AIO_SUCCESS)
{
    AIOBOARDLIST brd;
```



```

DlCallbackIndex = 0;
DlCallbackEscape = 0;
DlCallbackStarted = 0;
return(0);
}

/*****
 *
 * ValidPacket(BYTE *buf_to_rx,
 *             int *len_to_rx)
 *
 * Description:
 * This routine validates a response from the modem. It checks the
 * length, checksum, opcode and status.
 *
 * Input:
 *   buf_to_rx      - pointer to the response
 *   len_to_rx      - pointer to the total length of the message
 *
 * Output:
 *   len_to_rx      - changed to the real size of the message
 *
 * Returns:
 *   TRUE if packet is valid
 *
 *****/
int ValidExplicitPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    LroEspkt_t *msg;
    WORD frameCrc = 0;
    WORD slipCrc = 0;
    WORD frameLen = 0;
    WORD crcVal = 0xffff;

    msg = (LroEspkt_t *)buf_to_rx;
    frameLen = (msg->length) & 0x7fff;

    if ((*len_to_rx - sizeof(WORD)) == frameLen)
    {
        CMovB(&buf_to_rx[frameLen], (BYTE *)&frameCrc, sizeof(frameCrc));

        slipCrc = calcCrc(crcVal, buf_to_rx, frameLen);
        if (slipCrc == frameCrc)
        {
            return(1);
        }
    }

    return(0);
}

int ValidPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    unsigned short frameCrc=0; // CRC value contained in the frame
    unsigned short frameLen=0; // Length value contained in the frame
    unsigned short slipCrc=0; // CRC returned from calculation
    LroAsyncMsg_t *msg;

```

```

DlCallbackIndex = 0;
DlCallbackEscape = 0;
DlCallbackStarted = 0;
return(0);
}

/*****
 *
 * ValidPacket(BYTE *buf_to_rx,
 *             int *len_to_rx)
 *
 * Description:
 * This routine validates a response from the modem. It checks the
 * length, checksum, opcode and status.
 *
 * Input:
 *   buf_to_rx      - pointer to the response
 *   len_to_rx      - pointer to the total length of the message
 *
 * Output:
 *   len_to_rx      - changed to the real size of the message
 *
 * Returns:
 *   TRUE if packet is valid
 *
 *****/
int ValidExplicitPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    LroEspkt_t *msg;
    WORD frameCrc = 0;
    WORD slipCrc = 0;
    WORD frameLen = 0;
    WORD crcVal = 0xffff;

    msg = (LroEspkt_t *)buf_to_rx;
    frameLen = (msg->length) & 0x7fff;

    if ((*len_to_rx - sizeof(WORD)) == frameLen)
    {
        CMovB(&buf_to_rx[frameLen], (BYTE *)&frameCrc, sizeof(frameCrc));

        slipCrc = calcCrc(crcVal, buf_to_rx, frameLen);
        if (slipCrc == frameCrc)
        {
            return(1);
        }
    }

    return(0);
}

int ValidPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    unsigned short frameCrc=0; // CRC value contained in the frame
    unsigned short frameLen=0; // Length value contained in the frame
    unsigned short slipCrc=0; // CRC returned from calculation
    LroAsyncMsg_t *msg;

```

```

if (DioCallBackStarted == 0)
{
    if (value == END)
    {
        DioCallBackStarted = 1;
    }
    continue;
}

switch(value)
{
    /* if it's an END character then we're done with the packet */
    case END:
        /* We're done. */
        if (DioCallBackType == EXPLICIT_RECEIVE)
        {
            if (ValidExplicitPacket(DioCallBackBuffer, &DioCallBackIndex))
            {
                DioCallBack(DioCallBackBuffer + LRO_EXPLICIT_HDR_SIZE, ackIndex, 0);
            }
            else
            {
                DioCallBackIndex = 0;
                DioCallBackStarted = 0;
                DioCallBackEscape = 0;
                break;
            }
        }
        else
        {
            /* We're done */
            if (DioCallBackType == EXPLICIT_RECEIVE)
            {
                DioCallBack(DioCallBackBuffer + LRO_EXPLICIT_HDR_SIZE, 1);
            }
            else
            {
                DioCallBack(DioCallBackBuffer + RETURN_POINT + LRO_ASYNC_HDR_SIZE, 1);
            }
            DioCallBack = 0;
            return;
        }
        break;
    }

    LONG DPCNextRegistrationCheck;
    /*****
    *
    * DloMain(void *parm)
    *
    * Description:
    *   Main thread for modem handling.
    *   We first initialize the modem thru AIO. We then check the
    *   modem to see if we've received anything. If we did, we gather
    *   it until we get a carriage return. Then we check against expected
    *   responses. If we get a expected response, we set the appropriate
    *   event in the state machine. We then check for packet life
    *   and state machine timeouts. Otherwise, we sleep.
    *
    * Input:
    *   parm
    *
    * Output:
    *   nothing
    *
    * Returns:
    *   nothing
    *
    * - ignored
    */

```

```

*****
void DloMain(void *parm)
{
    LONG curticks, delta;
    LONG count, bytesRead;
    WORD state;
    BYTE value;
    int event, eventLen;
    char *pPtrAtBegin, *pPtrAtEnd;
    LONG extStatus;

    parm = parm;
    DloLastKnownTickCount = GetCurrentTime();

    while(!ExitingFlag)
    {
        delay(1000 / 6);

        count = 0;
        bytesRead = 0;
        if (AIOGetPortStatus(AIOPortHandle,
            &count, /* write count */
            0, /* write status */
            &bytesRead, /* read count */
            0, /* read status */
            &extStatus,
            0) == 0)
        {
            extStatus &= AIO_EXTSTA_DCD;
            if (extStatus != DloLastDCD)
            {
                if (!extStatus)
                    StateMachine(DLOE_DISCONN);
                DloLastDCD = extStatus;
            }

            UpdateModemLights(count, bytesRead, DloLastDCD);

            if (DloState == DLOS_IDLE)
            {
                if (DloPXMitCount)
                {
                    DloConn = DLO_CONN_PACKAGE;
                    StateMachine(DLOE_SEND);
                }
                else if (DloIXmitCount)
                {
                    DloConn = DLO_CONN_INET;
                    StateMachine(DLOE_SEND);
                }
            }

            curticks = GetCurrentTime();
            if (curticks < DloLastKnownTickCount)
            {
                delta = curticks + (0xffffffff - DloLastKnownTickCount);
            }
            else
            {
                delta = curticks - DloLastKnownTickCount;
            }
            DloLastKnownTickCount = curticks;

```

```

        if (DIOStats && curticks > DPCNextRegistrationCheck)
        {
            char buf[16];
            LONG hw;
            double key;
            CustVars* customPtr = (CustVars*)&(DIOStats->CustomVariableCount);

            LONG rxFreq = customPtr->CustomVariable[1] / 10;
            DPCSetMaxConnections((LONG*)&key);
            if (strcmp(DloCfg.base_license, "Helius, Inc.", 8) == 0)
            {
                DPCNextRegistrationCheck = (LONG)(-1);
                goto skipRegistrationCheck;
            }

            /* get hardware serial number */
            *buf = 0;
            DIOGetSN(buf);
            hw = strtoul(buf, 0, 10);
            if (hw == 0)
            {
                ConsolePrintf("\r\nDPCAgent: could not obtain hardware
                    serial number of DPC card\n");
                goto disableDPCAgent;
            }

            /* compute the registration key */
            if (DebugFlag == 0x98)
            {
                printf("\rRegCheck: %e\n", key);
                HexAsciiDump((void*)&key, sizeof(key));
            }
            key *= hw + DPC_IP_Address;
            if (DebugFlag == 0x98)
            {
                printf("\rRegCheck: %e %d %08x\n",
                    key, hw, DPC_IP_Address);
                HexAsciiDump((void*)&key, sizeof(key));
            }
            if (key == *(double*)&DloCfg.key)
                DPCNextRegistrationCheck = curticks + 131072; /*
                    120 minutes */
            else
            {
                ConsolePrintf("\r\nDPCAgent: detected a bad regi
                    stration key\n");

                disableDPCAgent:
                DPCMaxConnections = 3;
                RingTheBell();
                if (DPCNextRegistrationCheck)
                {
                    DloCfg.gateway_address = 0;
                    StateMachine(DLOE_TIMEOUT);
                    DPCNextRegistrationCheck = curticks + 2185; /*
                        2 minutes */
                }
                else
                {
                    DPCNextRegistrationCheck = curticks + 131072;
                }
            }
            if ((DPCNextRegistrationCheck & 0xffffffff) == 0xffffffff)
            {
                DPCNextRegistrationCheck += 17; /* wrap */
            }

            skipRegistrationCheck:
            if (AIOPortHandle >= 0)
            {

```

```

    for (;;)
    {
        || count == 0)

        break;
        AIOReadData(AIOPortHandle, &value, 1, &bytesRead);
        if (DebugFlag && DloState != DLOS_CONN)
            putchar(value);
        if (DloRcvCount < DLOBUFSIZE)
        {
            DloRcvBuffer[DloRcvIndex] = value;
            DloRcvIndex++;
            if (DloRcvIndex >= DLOBUFSIZE)
                DloRcvIndex = 0;
            DloRcvBuffer[DloRcvIndex] = 0;
            DloRcvCount++;
        }
        if (value != '\n' && value != '\r')
        {
            memcpy(DloCommandBuffer, DloCommandBuffer + 1, DLOCMDBUFSIZE - 1);
            DloCommandBuffer[DLOCMDBUFSIZE - 1] = value;
            if (DloCommandIndex < DLOCMDBUFSIZE)
                DloCommandIndex++;
        }
        if (value != '\r')
            continue;
        pPtrAtBegin = DloCommandBuffer + DLOCMDBUFSIZE - eventLen;
        if (strlen(pPtrAtBegin, DloCompareString[DLOE_CONNECT])
            DloCommandIndex;
        ONNECT], eventLen) == 0)
        {
            ntLen, DloCommandIndex - eventLen);
            n + 1] = 0;
        }
        else
        {
            for (event = 0; event < DLOENUM; event++)
            {
                if (event == DLOE_CONNECT)
                    continue;
                eventLen = CStrLen(DloCompareString[event], DloCommandBuffer + D
                if (strcmp(pPtrAtEnd, DloCompareString[event], eventLen) != 0)
                {
                    DloFlushReceive();
                    StateMachine(DLOE_CONNECT);
                }
            }
            if (event == DLOE_CONNECT)
                continue;
            eventLen = CStrLen(DloCompareString[event], DloCommandBuffer + D
            if (strcmp(pPtrAtEnd, DloCompareString[event], eventLen) != 0)
            {
                DloFlushReceive();
                StateMachine(DLOE_CONNECT);
            }
        }
    }
}

```

Thu Jul 17 14:46:11 1997

dloc

Page 55

00000000

```
if (delta >= DlocCallBackTimeout)
{
    /* Timeout!!! Call routine with
    DlocCallBackWait = 0;
    DlocCallBack(DlocCallBackBuffer, D
    DlocCallBack = 0;
    }
    else
        DlocCallBackTimeout -= delta;
    /* Read any available modem characters f
    DlocCallBackRead();
    }
    )
    if (AIOPortHandle >= 0)
        AIOReleasePort(AIOPortHandle);
    DPCModemPID = 0;
}
```

page ,130

page

```
*****\
; BEGIN_MANUAL_ENTRY( History, DPC/HISTORY )
;
; DirecPC Driver Code for NetWare 386.
; This driver must be loaded after MSM.NLM and ETHERNET.NLM.
;
; Written by:   DFS
; Date:        November, 1995
;
; *****
; History Log:
; version 0.0 - First demo version
; version 0.2 - (3-15-96) Put SignalQuality and RxFreq into stats table
; version 0.3 - (4-09-96) Fixed CloseChannel so that DPCAGENT.NLM
;                  can be properly reloaded.
; version 0.4 - (4-16-96) Added code to send turbo internet packets
;                  up to the ethertsm instead of DPCAGENT
; version 0.5 - (4-24-96) Adjusted RpacketOffset 2 bytes into
;                  envelope in order to build 14 byte ethernet header
;                  over DPC's 12 byte header to turbo internet packets
; version 0.6 - (5-16-96) Completion of Phase 2
;                  Debug screen only active if -DEBUG on the command line
; version 1.00 - (6-11-96)
; version 1.01 - (6-17-96) Added check to end of DriverInit for locked adapter
;                  Added Agent Register call to DriverManagement so that
;                  we can call agent when we're being removed, allowing
;                  agent to go into idle state until we're loaded again.
; version 1.02 - (6-20-96)
; version 1.03 - (6-24-96) Added Freq= to command line.
;                  Fixed packet too big bug.
; version 1.04 - (6-25-96) Really Fixed packet too big bug.
; version 1.05 - (6-28-96) Enhanced dpcagent.nlm
; version 1.06 - (7-10-96) Reduced MLIDMaximumSize so Netscape wouldn't overflow
;                  when tunneling enabled.
; version 1.07 - (7-13-96) Stopped calling SendComplete from DriverSend
;                  Added DriverTCBComplete to be obtained from IOCTL
;                  Bumped TxFreeCount to 32
; version 1.09a - (8-28-96) Reduced max packet size if we couldn't hook bind event.
; version 1.20 - 2-12-96
;                  Returned ECB allocated by FastProcessGetRCB
;                  Changed call to DataLinkRxFrame to DPCRxFrame
;                  Changed copyright string
;
; END_MANUAL_ENTRY
; *****
;
; name DPC
; title DirecPC LAN Driver (HSM Version)
; subttl -- Structures and Equate Values --
; page
;
; include driver.inc
; subttl -- DirecPC Specific Equates --
```

```
*****\
; *****
; Equates.
; *****
; subttl -- DirecPC Specific Structures --
; page
;
; MLID_MAJOR_VERSION equ 01
; MLID_MINOR_VERSION equ 20
;
; TRUE equ 01
; FALSE equ 00
;
; TIMESTAMP equ FALSE
; TIMESTAMP_BUFFER_SIZE equ (256 * 4)
;
; INT_3 equ int 3
;
; EVENT_PROTOCOL_BIND equ 33
; EVENT_PROTOCOL_UNBIND equ 34
;
; ; Status Register bit masks
;
; STAT_BUSY equ 01
; STAT_ERROR equ 02
;
; STAT_RX_INT equ 04h
; STAT_NO_RBD equ 08h
;
; STAT_CNTL_INT equ 10h
; STAT_SOUTPUT equ 20h
;
; ; Control Register bit masks
;
; CNTL_CPU_EN equ 01h
; CNTL_RX_EN equ 02h
; CNTL_INT_EN equ 04h
; CNTL_CHAN_ATT equ 08h
; CNTL_SINGL_INT equ 10h
; CNTL_15_INT equ F0h
; CNTL_AUTO_INC equ 100h
; CNTL_MRESET equ 200h
; CNTL_FORCEINT equ 400h
; CNTL_SOUTPUT equ 800h
; CNTL_IRQ3 equ 0000h
; CNTL_IRQ4 equ 2000h
; CNTL_IRQ5 equ 4000h
; CNTL_IRQ9 equ 6000h
; CNTL_IRQ10 equ 8000h
; CNTL_IRQ11 equ 0A000h
; CNTL_IRQ12 equ 0C000h
; CNTL_IRQ15 equ 0E000h
;
; ; Synthesizer chip types
;
; INVALID_TUNER equ Offh
; SHARP equ 00h
; ALPS equ 08h
; PANASONIC equ 20h
; SHARP_CUSTOM equ 28h
;
; ; Define State Numbers (informational only)
```

```
/* Adapter is rxing data */
/* One or more errors listed below: */
/* framing,CRC,Abort,Alignment,DES */
/* Rx interrupt */
/* RBD not ready (E bit wasn't cleared) */
/* by software */
/* Controller Interrupt-RISC handshake */
/* From CNTL Spare Output (0x800) */
```

```
/* RISC CPU Enable */
/* Rx Enable */
/* Interrupt Enable */
/* Another handshake with RISC CPU */
/* Bits 4-7: # of buffers per Rx Int */
/* (from 1 to 15) */
/* Auto Inc Message RAM Pointer */
/* Soft Reset and Clear All Registers */
/* 'Cause Interrupt' bit */
/* Spare Output to STAT Bit 5 (0x20) */
/* Bits 13-15: IRQ selection */
```

```
/* not supported */
```

```

;
INIT equ 0
SYNTH_PRGM equ 1
ACQ_PD_DELAY equ 2
ACQ_PD equ 3
ENABLE_BTR equ 4
START_SEARCH_FOR_FEC equ 5
CHECK_FOR_FEC_LOCK equ 6
SET_OTHER_MODE equ 7
TRACKING equ 8
POINTING_ACQ equ 9
POINTING_TRACKING equ 10
HALT equ 11

; New Stuff DBS
; demod command definitions
;
ACQUIRE_MODE equ 0
HALT_MODE equ 2
BUSY_MODE equ 3
POINTING_MODE equ 4

; /** start a new acquisition
; /** Do nothing
; /** Trying to acquire
; /** Special test mode
;

DEFAULT_RX_FREQ equ 1330
BIT_OFF equ 00h

; BtrControlAddr bits - write register 0
;
FREQ_PWR_MASK equ 0E0h
PHASE_PWR_MASK equ 07h
BTR_SENSE_MASK equ 08h
BTR_ERR_ENA_MASK equ 10h
FREQ_PWR_OFFSET equ 20h
PHASE_PWR_OFFSET equ 01h

; AfcControlAddr bits - write register 1
;
SWP_ENA_MASK equ 01h
SWEEP_DIR_SENSE_MASK equ 08h
AFC_SENSE_MASK equ 10h
EXT_INT_MASK equ 20h
BPSK_MASK equ 40h
ROM_ENA_MASK equ 80h
SQF_PEAK_EN_MASK equ 02h

; BtDetControlAddr bits - write register 2
;
SOFT_THRS_MASK equ 1Fh
DECODER_INFC_SEL_MASK equ 80h
VIT_SEQ_MASK equ 40h
SOFT_THRS_OFFSET equ 01h

; AgcFirControlAddr bits - write register 3
;
AGC_REF_MASK equ 1Fh
AGC_SENSE_MASK equ 40h
FIR_BYPASS_MASK equ 80h
SWAP_IQ_MASK equ 20h
AGC_REF_OFFSET equ 01h

; CrLkControlAddr bits - write register B
;
CRLK_DET_PWR_MASK equ 07h
CRLK_FC_MASK equ 38h
CRLK_GAIN_MASK equ C0h

; SynthSerControlAddr bits - write register C
;
SDATA_MASK equ 01h
SCLK_MASK equ 02h
SENA_MASK equ 04h
MODE_MASK equ 08h
CRL_ACC_ENABLE equ 10h
DEPUNC_BYPASS_MASK equ 20h
RESET_FEC_ACQ_MASK equ 40h
RESET_BTR_ACC_MASK equ 80h

; DaAdOffsetControlAddr bits - write register E
;
I_CHANNEL_OFFSET equ 01h
CRL_ERROR_OFFSET equ 08h
BTR_ERROR_OFFSET equ 40h
;
;
;
SYNTH_LOCK_MASK equ 01h
CRL_LOCK_MASK equ 02h
SWEEPING_MASK equ 04h
DIR_MASK equ 08h
FEC_LOCK_MASK equ 10h
TUNER_TYPE_MASK equ 20h
TUNER_TYPE_2_MASK equ 08h
VENDOR_ID_MASK equ 0F0h

; /** Frequency offsets */
;
PLUS_OFFSET equ 40
ZERO_OFFSET equ 0
MINUS_OFFSET equ -40
OFFSET_THRESHOLD equ 1152
FREQ_BASE equ 9011
K_TRACK equ 1736
K_REACQ equ 29622
NOM_COUNT_TRACK equ 48761
NOM_COUNT_REACQ equ 19432
SYNTH_CHANNEL_SIZE equ 3645
SYNTH_RATIO equ 64
SYNTH_CHANNELS_PER_STEP equ 40
SYNTH_FIRST_CHANNEL equ 3788

BPSK equ BPSK_MASK OR ROM_ENA_MASK
;
; /** Possible Viterbi modes */
;
LOWRATE equ 0
HIGHRATE equ 1
;
; /** demod status states */
;
UNLOCKED equ 0
LOCKED equ 1
;
; Configuration equates
;
RBD_BASE_ADDR equ 0a000h
ADAP_RBD_NUM equ 128
RBD_BUFFER_SIZE equ 1024
LOCAL_BUF_NUM equ 400

```

```
; RBD status bits.
;
; FRAMING_ERR      equ 0001h      ; Framing error
; CRC_ERR          equ 0002h      ; CRC error
; ABORT            equ 0004h      ; Abort
; ALIGN_ERR        equ 0008h      ; Alignment error
; DES_ERR          equ 0010h      ; DES Error
; SOF_BIT          equ 0020h      ; Start of Frame(not used)
; EOF_BIT          equ 0040h      ; End of Frame
; OVERRUN_ERR      equ 0080h      ; Frame Overrun bit
; EMPTY           equ 8000h      ; if reset, buffer may be used
; STATUS_ERROR     equ          FRAMING_ERR OR CRC_ERR OR ABORT OR ALIGN_ERR OR
; DES_ERR OR OVERRUN_ERR
```

```
DebugMessage macro mask, message
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (2 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage1 macro mask, message, parm0
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (3 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage2 macro mask, message, parm0, parm1
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (4 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage3 macro mask, message, parm0, parm1, parm2
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm2
    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (5 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
    DebugMessageExit:
    endm
```

```
DebugMessage4 macro mask, message, parm0, parm1, parm2, parm3
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx

    push parm3
    push parm2
    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (6 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

DebugMessageExit:

endm

DebugMessage5 macro mask, message, parm0, parm1, parm2, parm3, parm4

local DebugMessageExit

test DebugMask, mask

je DebugMessageExit

push eax

push ecx

push edx

push parm4

push parm3

push parm2

push parm1

push parm0

push offset message

push DPCScreen

call OutputToScreen

lea esp, [esp + (7 * 4)]

pop edx

pop ecx

pop eax

DebugMessageExit:

endm

DebugMessage6 macro mask, message, parm0, parm1, parm2, parm3, parm4, parm5

local DebugMessageExit

test DebugMask, mask

je DebugMessageExit

push eax

push ecx

push edx

xor eax, eax

mov al, parm5

push eax

mov al, parm4

push eax

mov al, parm3

push eax

mov al, parm2

push eax

mov al, parm1

push eax

mov al, parm0

push eax

push offset message

push DPCScreen

call OutputToScreen

lea esp, [esp + (8 * 4)]

pop edx

pop ecx

pop eax

DebugMessageExit:

endm

SLOW

macro

push eax

in al, 61h

in al, 61h

in al, 61h

pop eax

endm

BufferStruct struc

BufPtr

dd 0

dd 0

ends

BufferStruct

ends

LAST_EOF

equ 80000000h

MAX_APPL_RBD

equ 350

MAX_CHAN

equ 10

RBD_NOT_USED

equ -1

MAX_CONF_ADDR

equ 8

MAX_ADDR

equ 16

RX_CNTL struc

RxChannel

dd 0

RxESR

dd 0

ends

RX_CNTL

ends

RX_FLAG_STATS_ONLY

equ 1

ChannelConfig struc

CfgChannel

dd 0

CfgESR

dd 0

CfgNumAddresses

dd 0

CfgAddress

db 6 dup (0)

CfgGroupKey

db 8 dup (0)

CfgElementKey

db 8 dup (0)

ends

ChannelConfig

ends

FilterStruct struc

FilterAddress

db 6 dup (0)

FilterChannel

db 2 dup (0)

FilterCmdBlkIndex

dd 0

FilterTotalCount

dd 0

FilterSeqCount

dd 0

FilterSeqNum

dd 0

ends

FilterStruct

ends

; Pointer to buffer

; Size of buffer

; or with DataSize

; Channel ID

; ESR to call when a packet

; is received.

; max # of buffers used

; Number of filter addrs

; that is to follow

; list of addrs

; Group keys

; Element keys

; 48 bit address

; Align next field

; Channel ID

; RISC SW cmd blk index

; # of frames rxed

; # of out-of-seq frames

; next seq expected

; dw 0

; dw 0

; dw 0

; db 6 dup (0)

; db 2 dup (0)

; dd 0

; dd 0

; dd 0

; dd 0

; dd 0

; dw 6 dup (0)

; dw 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

; db 8 dup (0)

Thu Jul 17 14:46:01 1997			Page 11			Page 12		
dpc.386			dpc.386			dpc.386		
EblkBusyFlags	db	32 dup (0)	AveLargerRx	dd	0			
BufferCount	dd	0	MipsRxEnables	dd	0			
WatchBufferCount	dd	0	s the RSW has gotten a rx	dd	0			
WatchOldRejected	dd	0	MipsRxdisables	dd	0			
;;LocalMipsStats	db	(size StatsBlk) dup (0)	commands.	dd	0			
AgentSendRoutine	dd	0	MipsFramesAccepted	dd	0			
AgentRemoveRoutine	dd	0	MipsNoFilterMatch	dd	0			
GotInterrupt	dd	0	to no filter match.	dd	0			
IOEnableValue	dd	0	MipsZeroAddrFrames	dd	0			
;			MipsDisabledRejeced	dd	0			
;			be rejected while zeroed.	dd	0			
;			MipsFullBufsRejec	dd	0			
;			use the	dd	0			
;);	dd	0			
;); adapter was close to running out of buffers (RBDS	dd	0			
StatisticsVersion	db	03, 00	VectorToTheStrings	dd	offset DiagnosticsStrings			
GenericVariableCount	dw	(RxAbortFrameAlignment + 4 - TotalTxPacketCount)	TotalLargeRx	dd	0			
NotSupportedMask0	dd	1111101111001000000000000000000011b	AligndEndVA	db	((4 - offset (AligndEndVA and 3)) and 3) + 4 dup			
			(?) ; Align 4 for MOVSD	dd				
TotalTxPacketCount	dd	0	DriverAdapterDataSpace	ends				
TotalRxPacketCount	dd	0	IORbdsBase	equ	IOStatus			
NoECBAvailCount	dd	0	IOtuningLowAddr	equ	IOCountNomLowAddr			
PacketTxTooBigCount	dd	0	IOtuningHighAddr	equ	IOCountNomHighAddr			
PacketTxTooSmallCount	dd	0	IOrelSgfAddr	equ	IOCountDeltaAddr			
PacketRxTooBigCount	dd	0	IOMaxSgfAddr	equ	IOSweepRateAddr			
PacketRxTooSmallCount	dd	0	IOStatusAddr	equ	IODaOffsetControlAddr			
PacketTxMiscErrorCount	dd	0	subttl	-- OSDATA Data Segment --				
PacketRxMiscErrorCount	dd	0	page	assume cs: OSCODE, ds: OSDATA, es: OSDATA, ss: OSDATA				
RetryTxCount	dd	0	;	*****				
ChecksumErrorCount	dd	0	;	*****				
HardwareRxMismatchCount	dd	0	;	*****				
TotalTxOKByteCountLow	dd	0	;	*****				
TotalTxOKByteCountHigh	dd	0	;	*****				
TotalRxOKByteCountLow	dd	0	;	*****				
TotalRxOKByteCountHigh	dd	0	;	*****				
TotalGroupAddrTxCount	dd	0	;	*****				
TotalGroupAddrRxCount	dd	0	;	*****				
AdapterResetCount	dd	0	;	*****				
AdapterOpTimeStamp	dd	0	;	*****				
QDepth	dd	0	;	*****				
TxOKSingleCollision	dd	0	;	*****				
TxOKMultipleCollisions	dd	0	;	*****				
TxOKButDeferred	dd	0	;	*****				
TxAbortLateCollision	dd	0	;	*****				
TxAbortExcessCollisions	dd	0	;	*****				
TxAbortCarrierSense	dd	0	;	*****				
TxAbortExDeferral	dd	0	;	*****				
RxAbortFrameAlignment	dd	0	;	*****				
CustomVariableCount	dw	(MipsFullBufsRejec + 4 - SignalQuality)	DiagnosicsStrings	dw	(EndOfStrings-DiagnosticsStrings)			
SignalQuality	dd	0	;	*****				
from adapter	dd	0	;	*****				
RxFreq	dd	0	;	*****				
ncy	dd	0	;	*****				
LargestRx	dd	0	;	*****				
NumberLargeRx	dd	0	;	*****				

```
db 'Mips Rx disable commands', 0
db 'Mips frames accepted', 0
db 'Mips frames rejected from no filter match', 0
db 'Mips frames rejected from zero address', 0
db 'Mips frames rejected from adapter disabled', 0
db 'Mips frames rejected from low buffer pool', 0

db 0,0

EndOfStrings equ $
;*****
;
; Driver Parameter Block to pass to MSM.
;*****
;*****
; align 4
DriverParameterBlock
DriverParameterSize dd DriverParameterBlockSize
DriverStackPointer dd 0
DriverModuleHandle dd 0
DriverBoardPointer dd 0
DriverAdapterPointer dd 0
DriverConfigTemplatePtr dd DriverConfigTemplate
DriverFirmwareSize dd 0
DriverFirmwareBuffer dd 0
DriverNumKeywords dd 2
DriverKeywordText dd DPCKeywordText
DriverKeywordTextLen dd DPCKTextLen
DriverProcessKeywordTab dd DPCKProcessKeywordTab
DriverAdapterDataSpaceSize dd SIZE DriverAdapterDataSpace
DriverAdapterTemplate dd DriverAdapterDataSpaceTemplate
DriverStatisticsTable dd StatisticsVersion
DriverEndOfChainFlag dd 0
DriverSendWantsECBs dd 0
DriverMaxMulticast dd -1
DriverNeedsBelow16Meg dd 0
DriverAESPtr dd 0
DriverCallBackPtr dd offset DriverCallBack
DriverISRPtr dd offset DriverISR
DriverMulticastChangePtr dd offset DriverMulticastChange
DriverPollPtr dd 0
DriverResetPtr dd offset DriverReset
DriverSendPtr dd offset DriverSend
DriverShutdownPtr dd offset DriverShutdown
DriverTxTimeoutPtr dd 0
DriverPromiscuousChangePtr dd offset DriverPromiscuousChange
DriverStatisticsChangePtr dd offset RefreshMipsStats
DriverRxLookAheadChangePtr dd 0
DriverManagementPtr dd offset DriverManagement
DriverEnableInterruptPtr dd offset DriverEnableInterrupt
DriverDisableInterruptPtr dd offset DriverDisableInterrupt

DriverParameterBlockSize equ $ - DriverParameterBlock
;*****
;
; Driver Management Dispatch Jump Table.
;*****
;*****
; ManagementJumpTable label dword
; dd offset GetMipsStats
; dd offset OpenChannel
; dd offset CloseChannel
```

```
dd offset GetSN
dd offset SignText
dd offset AddrAddress
dd offset DeleteAddress
dd offset RegisterAgentSendRoutine
dd offset BadParametersExit
dd offset RegisterAgent
dd offset ReturnTCBCompleteRoutine
LastManagementFunction equ (($ - ManagementJumpTable) / 4) - 1
;*****
;
; Copy of Virtual Adapter Data area to be copied at
; initialization.
;*****
;*****
; DriverAdapterDataSpaceTemplate DriverAdapterDataSpace <>
DriverConfigTemplate db 'HardwareDriverMLID
Signature
db 01 ; [ebx].MLIDCFG.MajorVersion
db 12 ; [ebx].MLIDCFG.MinorVersion
db 6 dup (0Fh) ; [ebx].MLIDNodeAddress
dw 0010010001001001b ; [ebx].MLIDModeFlags
dw 0000 ; [ebx].MLIDBoardNumber
dw 0000 ; [ebx].MLIDBoardInstance
dd 00000000 ; [ebx].MLIDMaxImumSize
dd 00000000 ; [ebx].MLIDMaxRecvSize
dd 00000000 ; [ebx].MLIDRecvSize
dd 00000000 ; [ebx].MLIDCardName
dd DriverNICShortName ; [ebx].MLIDShortName
dd 00000000 ; [ebx].MLIDFrameType
dw 0000 ; [ebx].MLIDReserved0
dw 0000 ; [ebx].MLIDFrameID
dw 0001 ; [ebx].MLIDTransportTime
dd 00000000 ; [ebx].MLIDRouteHandler
dw 10 ; [ebx].MLIDLineSpeed
dw 0000 ; [ebx].MLIDLookAheadSize
db 8 dup (00h) ; [ebx].MLIDReserved
MLID_MAJOR_VERSION ; [ebx].MLIDMajor
MLID_MINOR_VERSION ; [ebx].MLIDMinor
rVersion db
rVersion dw 0010b ; [ebx].MLIDFlags
dw 0010 ; [ebx].MLIDSendRetries
dd 00000000 ; [ebx].MLIDLink
dw 0000 ; [ebx].MLIDSharingFlags
dw 0000 ; [ebx].MLIDSlot
dw 02c0h, 40h, 0, 0 ; [ebx].MLIDIOPortsAndLengths
dd 00000000 ; [ebx].MLIDMemoryDecoded0
dw 0000 ; [ebx].MLIDLength0
dd 00000000 ; [ebx].MLIDMemoryDecoded1
dw 0000 ; [ebx].MLIDLength1
db 03, 0Fh ; [ebx].MLIDInterrupt
db 0Fh, 0Fh ; [ebx].MLIDMAUsage
dd 00000000 ; [ebx].MLIDResourceTag
dd 00000000 ; [ebx].MLIDConfiguration
dd 00000000 ; [ebx].MLIDCommandString
db 18 dup (0) ; [ebx].MLIDLogicalName
dd 00000000 ; [ebx].MLIDLinearMemory0
dd 00000000 ; [ebx].MLIDLinearMemory1
dw 0000 ; [ebx].MLIDChannelNumber
db 6 dup (0) ; [ebx].MLIDIOReserved
Message DriverNICShortName, 'DPC'
```

```
;*****
; Parameters required by ParseDriverParameters.
;*****
DebugText      db '-DEBUG' ; Break into debugger keyword.
DebugTextLen   equ $-DebugText OR T_STRING
FreqText       db 'FREQ' ; Set Rx/Freq
FreqTextLen    equ $-FreqText OR T_NUMBER
               dd 0 ; min
               dd 10000 ; max
               dd 0
               dd 0
               dd 0

DPCKeywordText dd DebugText ; First Keyword.
               dd FreqText ; Second Keyword.
DPCTextLen     dd DebugTextLen ; First Keywords length.
               dd FreqTextLen ; Second Keywords length.

DPCProcessKeywordTab dd DebugRoutine ; First Keyword routine.
                   dd FreqRoutine ; Second Keyword routine.
                   dd DEFAULT_RX_FREQ
                   dd 12, 100h, 140h, 180h, 1c0h
                   dd 200h, 240h, 280h, 2c0h
                   dd 300h, 340h, 380h, 3c0h
                   dd 8, 3, 4, 5, 9, 10, 11, 12, 15

Interrupt0Data
;
IOPort0Data
;
AdapterOptions AdapterOptionDefinitionStructure <, IOPort0Data, ....., Interrupt0Data>

SearchTbl
dd MINUS_OFFSET
dd ZERO_OFFSET
dd PLUS_OFFSET

StateTbl
dd offset InitState
dd offset SynthPrgrmState
dd offset AcqPDDelayState
dd offset AcqPDSState
dd offset EnableBTRState
dd offset StartSearchForFECState
dd offset CheckForFEClockState
dd offset SetOtherModeState
dd offset TrackingState
dd offset PointingAcquisitionState
dd offset PointingTrackingState
dd offset HaltState

MajorVer       dd 0
MinorVer       dd 0
Year           dd 0
Month          dd 0
Day            dd 0
ScreenRTag     dd 0
DPCScreen      dd 0

OurAdapterDataSpace dd 0
;*****
; Mipx code.
; Contains array MipxCode and dword MipxCodeSize.
;*****
include mips.dat
;*****
; Message equates.
;*****
CR equ 13
LF equ 10
;*****
; Run-time error message strings.
; DPC error messages.
;*****
TransmitTimeoutMsg db 66, 00, 'The cable might be disconnected on the board.', CR, LF, 0
DMACompleteError db 200, 00, 'The board's DMA did not complete the write.', CR, LF, 0
;*****
; InitNIC error message strings.
;*****
ErrorAllocatingRTagMessage db 72, 00, 'A resource tag is unavailable.', CR, LF, 0
NICNotInSlotMessage db 50, 00, 'The board cannot be found.', CR, LF, 0
PortFailMessage db 54, 00, 'The board did not respond to the initialization command.', CR, LF, 0
NICIn8BitSlotMessage db 223, 00, 'The board must be placed in a 16-bit slot.', CR, LF, 0
NICIsNE1000Message db 224, 00, 'This board is configured as an NE1000.', CR, LF, 0
NICBadConfiguration db 80, 00, 'The board could not be configured.', CR, LF, 0
BadISRMsg db 207, 00, 'Unable to set ISR for test ISR.', CR, LF, 0
NoInterruptMsg db 208, 00, 'Interrupt is non functional. Try using another interrupt.', CR, LF, 0
ErrorAllocatingMemoryMessage db 73, 00, 'Unable to allocate memory.', CR, LF, 0
MsgIOSetFailed db 200, 00, 'Unable to set the spare output. Choose another base I/O port.', CR, LF, 0
MsgIOClearFailed db 201, 00, 'Unable to clear the spare output. Choose another base I/O port.', CR, LF, 0
MsgBadRAM db 51, 00, 'Board RAM failed the memory test.', CR, LF, 0
LockedAdapterMsg db 202, 00, 'Unable to start the adapter. Power down system and reboot.', CR, LF, 0
TimerDesc db "DPC Timer", 0
EventTagMessage db "DPC Events", 0
InterruptRTagMessage db "DPC Test ISR", 0
LastDebugMessage dd
```

```

LastSignalStrength      dd      0

SignalStrengthMsg      db
HaltStateMsg           db
PointingTrackStateMsg  db
PointingAcqStateMsg    db
TrackingStateMsg       db
SetOtherModeStateMsg  db
CheckForFECLockStateMsg db
StartSearchForFECMsg  db
EnableBTRMsg           db
AcqPDMsg               db
AcqPDDelayMsg          db
SynthPrgrmMsg          db
InitStateTrackMsg      db
InitStateNoTrackMsg    db
, 0
ChannelNumberMsg       db
SharpTunerMsg           db
PanasonicTunerMsg      db
SharpCustomTunerMsg    db
ScreenResourceName     db
ScreenName              db
DebugRBDReceived       db
DebugRBDSize           db
DebugScreenHeader      db
AdapterRBDMsg          db
FilterAddrMsg          db
2.2x %2.2x", CR, LF, 0
FilterRBDsMaxed        db
wanted %d", CR, LF, 0
FilterRBDLen           db
e = %d, headers = %d", CR, LF, 0
FilterCopy              db
ent %d", CR, LF, 0
FilterNone              db
%02.2x %02.2x %02.2x", CR, LF, 0
ISREnterMsg            db
ISRExitMsg             db
F, 0
FramingErrMsg          db
AbortMsg               db
AlignErrMsg            db
OverrunErrMsg          db
DESERtMsg              db
CRCERtMsg              db
NoCECMsg               db
ReturnMsg              db
AddAddrMsg             db
DebugInitOK            db
IPName                 db
NLName                 db
;
; If command line contains "-DEBUG", DebugMask will be set to DEBUG_MASK
;
;
DEBBUG_ISR             equ      0001h
DEBBUG_ISR_ALL         equ      0002h
DEBBUG_IOCTL           equ      0004h

DEBBUG_MASK            equ
DebugMask              dd      0,

align 4

```

```

SEND_STACK_SIZE equ 4000h

SendStackEnd          dd
SendStackBegin        dd      0
; ; OldSendStack      dd      0
;
; public DPCRxFrame
; DPCRxFrame          dd      0
;
if TIMESTAMP
; public DPCRTB
; DPCRTB              db      TIMESTAMP_BUFFER_SIZE dup (0)
;
; timestamp_begin dd      0
; timestamp_end   dd      0
; timestamp_index dd      0
;
; DPCDead          dd      0
; endif
;
; OSDATA          ends
; subttl          -- DriverChangeMulticast --
; page
; OSCODE          segment er public 'CODE'
;
; DebugRoutine     proc
; mov              DebugMask, DEBUG_MASK
; ret
;
; DebugRoutine     endp
;
; FreqRoutine       proc
; mov              GlobalRxFreq, eax
; ret
;
; FreqRoutine       endp
;
; *****
; ; BEGIN_MANUAL_ENTRY( DriverManagement, DPC/API/MANAGE )
; ;
; ; Name:           DriverManagement
; ;
; ; Description:    This routine will handle any DirecPC specific
; ;                commands called by a remote nlm.
; ;
; ; On Entry:       EAX      N/A
; ;                EBX      @ FrameDataSpace
; ;                ECX      N/A
; ;                EDX      N/A
; ;                EBP      @ Adapter Data Space
; ;                ESI      @ ECB
; ;                EDI      N/A
; ;
; ; Note:           Interrupts are in any state.
; ;
; ; On Return:      EAX      0 if successful
; ;                EBX      Preserved
; ;                ECX      Destroyed
; ;                EDX      Destroyed
; ;                EBP      Preserved
; ;                ESI      Destroyed
; ;

```


WaitForCommandDone:

```

push    ecx
mov     eax, [ebp].TimerTag
push    eax
push    2
call    DelayMyself
add     esp, (2 * 4)
pop     ecx

cli
mov     edx, [ebp].IOMsgRamPtr
mov     eax, (18100h + (14 * size Eb1kStruct)) / 2
out     dx, ax

mov     edx, [ebp].IOMsgRam
in      ax, dx
sti
cmp     ax, 0eeh
je      SignTextError

cmp     ax, 11h
je      ReadyToGetSignature
dec     ecx
jne     WaitForCommandDone

```

ReadyToGetSignature:

```

mov     edi, [esi+8]
cli

```

```

mov     edx, [ebp].IOMsgRamPtr
mov     eax, 18790h / 2
out     dx, ax

```

```

mov     ecx, 4
mov     edx, [ebp].IOMsgRam

```

GetSignatureLoop:

```

in      ax, dx
mov     [edi], ax
add     edi, 2
dec     ecx
jne     GetSignatureLoop

```

```

sti
mov     dword ptr [esi+4], 8
xor     eax, eax
ret

```

SignTextError:

```

mov     eax, -1
ret

```

SignText

```

subttl  -- GetSN --
page

```

; *****\

; BEGIN_MANUAL_ENTRY(GetSN, DPC/API/GETSN)

; Name: GetSN

; Description: This routine is called by DriverManagement to
; return the adapters serial number.

;

; On Entry:

```

EAX    N/A
EBX    Frame Data Space
ECX    N/A
EDX    N/A
EBP    Adapter Data Space
ESI    ECB
EDI    N/A

```

; Note: Interrupts are in any state.

```

On Return:
EAX    Destroyed
EBX    Preserved
ECX    Destroyed
EDX    Destroyed
EBP    Preserved
ESI    Preserved
EDI    Preserved

```

; Flags:

; Note: Interrupts preserved.

```

; Remarks: This routine is called by DriverManagement.
;           It is called at process time.

```

; See Also:

; END_MANUAL_ENTRY

; *****\

; public GetSN
; proc

mov esi, [esi].RPacketOffset

cli

```

mov     edx, [ebp].IOMsgRamPtr
mov     eax, 18760h / 2
out     dx, ax

```

```

mov     ecx, 3
mov     edx, [ebp].IOMsgRam

```

GetSNLoop:

```

in      ax, dx
mov     [esi], ax
add     esi, 2
dec     ecx
jne     GetSNLoop

```

```

sti
mov     [esi], cl
ret

```

```

GetSN    endp
subttl  -- CloseChannel --
page

```

; *****\

; BEGIN_MANUAL_ENTRY(CloseChannel, DPC/API/CLCHAN)

; Name: CloseChannel

; Description: This routine is called by DriverManagement to

```
; close the specified channel.
```

```
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  ECB
;            EDI  N/A
```

```
; Note:  Interrupts are in any state.
```

```
; On Return:  EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
```

```
; Flags:
```

```
; Note:  Interrupts preserved.
```

```
; Remarks:  This routine is called by DriverManagement.
;           It is called at process time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
; *****
```

```
public CloseChannel
CloseChannel proc
```

```
    mov     esi, [esi].RPacketOffset    ; ESI -> config structure
    mov     esi, [esi]                  ; ESI = channel
```

```
    cmp     esi, MAX_CHAN
    ja      CloseChannelError
    lea     edi, [ebp].RxControl[esi*8]
    cmp     [edi].RxChannel, RBD_NOT_USED
    je      CloseChannelError
```

```
    mov     [edi].RxChannel, RBD_NOT_USED
    mov     [edi].RXESR, 0
```

```
    mov     ecx, MAX_ADDR
    lea     edi, [ebp].Filter
```

```
CloseChannelCloseFilterLoop:
    cmp     [edi].FilterChannel, esi
    jne     CloseChannelCloseFilterNext
```

```
    mov     [edi].FilterChannel, RBD_NOT_USED
    mov     eax, [edi].FilterCmdblkIndex
    mov     [ebp].EblkBusyFlags[esi], 0
```

```
    push    ecx
    mov     ecx, size EblkStruct
    mul     ecx
    mov     edx, [ebp].IOMsgRamPtr
    add     eax, 18100h
```

```
    shr     eax, 1
    push    eax
    add     eax, 3
```

```
    out     dx, ax
```

```
    mov     edx, [ebp].IOMsgRam
    xor     eax, eax
    out     dx, ax
    out     dx, ax
    out     dx, ax
    out     dx, ax

    pop     eax
    pop     ecx
    mov     edx, [ebp].IOMsgRamPtr
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     eax, 1
    out     dx, ax
```

```
CloseChannelCloseFilterNext:
```

```
    add     edi, size FilterStruct
    dec     ecx
    jne     CloseChannelCloseFilterLoop
```

```
    xor     eax, eax
    ret
```

```
CloseChannelError:
```

```
    mov     eax, -1
    ret
```

```
CloseChannel endp
    subttl  -- Address --
    page
```

```
; *****
```

```
; BEGIN_MANUAL_ENTRY( Address, DPC/API/ADDRS )
```

```
; Name:      Address
```

```
; Description: This routine is called by DriverManagement to
;              add the address passed in.
```

```
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  ECB
;            EDI  N/A
```

```
; Note:  Interrupts are in any state.
```

```
; On Return:  EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
```

```
; Flags:
```

```
; Note:  Interrupts preserved.
```

```
; Remarks:  This routine is called by DriverManagement.
;           It is called at process time.
```

; See Also:

; END_MANUAL_ENTRY

; *****

```

;
; public AddAddress
; proc

```

```

mov     esi, [esi].RPacketOffset
mov     eax, [esi].CfgChannel
cmp     eax, MAX_CHAN
ja      OpenChannelError

lea     edi, [ebp].RxControl(eax*8)
cmp     [edi].RxChannel, RBD_NOT_USED
je      OpenChannelError

```

```

;
; ; Fall thru to AddAddr
;

```

```

AddAddress     endp
subttl  -- AddAddr --
page

```

; *****

; BEGIN_MANUAL_ENTRY(AddAddr, DPC/API/ADDADDR)

; Name: AddAddr

```

; Description: This routine is called by AddAddress and OpenChannel
;              to add the address to the adapter.

```

```

; On Entry:   EAX   N/A
;             EBX   Frame Data Space
;             ECX   N/A
;             EDX   N/A
;             EBP   Adapter Data Space
;             ESI   ECB
;             EDI   N/A
;
; Note:       Interrupts are in any state.

```

```

; On Return:  EAX   Destroyed
;             EBX   Preserved
;             ECX   Destroyed
;             EDX   Destroyed
;             EBP   Preserved
;             ESI   Preserved
;             EDI   Preserved

```

Flags:

Note: Interrupts preserved.

```

; Remarks:    This routine is called by AddAddress and OpenChannel.
;             It is called at process time.

```

; See Also:

; END_MANUAL_ENTRY

; *****

; ...

public AddAddr

AddAddr proc

```

mov     eax, dword ptr [esi].CfgAddress
DebugMessage1  DEBUG_IOCTL, AddAddrMsg, eax

```

```

mov     ecx, [esi].CfgNumAddresses

```

; ECX = Number 0

```

; Addrs
AddAddrLoop:

```

; First make sure this address is not a duplicate

```

;
; lea     edi, [ebp].Filter
; mov     edx, MAX_ADDR
;
; AddAddrDuplicateLoop:
; cmp     [edi].FilterChannel, RBD_NOT_USED
; je      AddAddrDuplicateNext
; mov     eax, dword ptr [edi].FilterAddress
; cmp     eax, dword ptr [esi].CfgAddress
; jne     AddAddrDuplicateNext
; mov     ax, word ptr [edi].FilterAddress+4
; cmp     ax, word ptr [esi].CfgAddress+4
; jne     AddAddrDuplicateNext

```

```

mov     eax, -1
ret

```

```

AddAddrDuplicateNext:
add     edi, size FilterStruct
dec     edx
jne     AddAddrDuplicateLoop

```

; Find an empty slot in the filter table

```

;
; lea     edi, [ebp].Filter
; xor     edx, edx
;
; AddAddrFindEmptyLoop:
; cmp     [edi].FilterChannel, RBD_NOT_USED
; je      AddAddrFindEmptyFound
; add     edi, size FilterStruct
; inc     edx
; cmp     edx, MAX_ADDR
; jnb     AddAddrFindEmptyLoop

```

```

mov     eax, -1
ret

```

```

AddAddrFindEmptyFound:
mov     eax, [esi].CfgChannel
mov     [edi].FilterChannel, eax
mov     eax, dword ptr [esi].CfgAddress
mov     ax, word ptr [edi].FilterAddress+4
mov     word ptr [edi].FilterAddress+4, ax
mov     [edi].FilterTotalCount, 0
mov     [edi].FilterSeqCount, 0
mov     [edi].FilterSeqNum, 0

```

```

mov     edx, 16
mov     eax, 31
test    [esi].CfgAddress+0, 02h
jnz     AddAddrFindEblkLoop
mov     edx, 2
mov     eax, 13

```

```

AddAddrFindEblkLoop:
    cmp     [ebp].EblkBusyFlags[edx], 0
    je      AddAddrFindEblkFound
    inc     edx
    cmp     edx, eax
    jbe     AddAddrFindEblkLoop
    mov     eax, -1
    ret

AddAddrFindEblkFound:
    mov     [ebp].FilterCmdIdx, 1
    mov     [edi].FilterCmdIdx, edx
    mov     eax, [edi].FilterChannel
    lea     edi, [ebp].Eblk
    mov     [edi].EblkCmd, 0
    mov     ax, word ptr [esi].CfgAddress
    xchg    ah, al
    word ptr [edi].EblkAddress, ax
    mov     ax, word ptr [esi].CfgAddress+2
    xchg    ah, al
    word ptr [edi].EblkAddress+2, ax
    mov     edx, 16
    AddAddrIsBypass
    mov     eax, dword ptr [esi].CfgGroupKey
    mov     dword ptr [edi].EblkGroupKey, eax
    mov     eax, dword ptr [esi].CfgGroupKey+4
    mov     dword ptr [edi].EblkGroupKey+4, eax
    mov     eax, dword ptr [esi].CfgElementKey
    mov     dword ptr [edi].EblkElemKey, eax
    mov     eax, dword ptr [esi].CfgElementKey+4
    mov     dword ptr [edi].EblkElemKey+4, eax

AddAddrIsBypass:
    pushfd
    cli
    push    esi
    mov     ecx, edx
    mov     ecx, size EblkStruct
    mul     ecx
    mov     edx, [ebp].IOMsgRamPtr
    add     eax, 18100h
    shr     eax, 1
    push    eax
    push    esi
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     ecx, size EblkStruct / 2
    mov     esi, edi
    cld
    AddAddrCopyEblk:
    lodsw
    out     dx, ax
    dec     ecx
    jne     AddAddrCopyEblk
    pop     esi
    pop     eax
    pop     ecx
    mov     edx, [ebp].IOMsgRamPtr
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     eax, 1

```

```

AddAddr     endp
subttl      -- DeleteAddress --
page
; *****
; BEGIN_MANUAL_ENTRY( DeleteAddress, DPC/API/DELADDR )
;
; Name:      DeleteAddress
;
; Description: This routine is called by DriverManagement to
;              delete the address passed in.
;
; On Entry:   EAX      N/A
;              EBX      Frame Data Space
;              ECX      N/A
;              EDX      N/A
;              EBP      Adapter Data Space
;              ESI      ECB
;              EDI      N/A
;
; Note:       Interrupts are in any state.
;
; On Return:  EAX      Destroyed
;              EBX      Preserved
;              ECX      Destroyed
;              EDX      Destroyed
;              EBP      Preserved
;              ESI      Preserved
;              EDI      Preserved
;
; Flags:
;
; Note:       Interrupts preserved.
;
; Remarks:    This routine is called by DriverManagement.
;              It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
; *****
;
; public DeleteAddress
; DeleteAddress proc
;
;     mov     esi, [esi].RPacketOffset      ; ESI -> config structure
;     mov     eax, [esi].CfgChannel
;     cmp     eax, MAX_CHAN
;     ja      DeleteAddressError
;
;     lea     edi, [ebp].RxControl[eax*8]
;     cmp     [edi].RxChannel, RBD_NOT_USED
;     je      DeleteAddressError
;
;     mov     ecx, [esi].CfgNumAddresses

```

```

cmp     ecx, MAX_CONF_ADDR
ja      DeleteAddressError

lea     ebx, [esi].CfgAddress
DeleteAddressLoop:

mov     ch, MAX_ADDR
lea     edi, [ebp].Filter
DeleteAddressFilterLoop:
mov     eax, dword ptr [ebx+0]
cmp     eax, dword ptr [edi].FilterAddress+0
jne     DeleteAddressNextFilter
mov     ax, word ptr [ebx+4]
cmp     ax, word ptr [edi].FilterAddress+4
jne     DeleteAddressNextFilter

mov     eax, [esi].CfgChannel
cmp     eax, [edi].FilterChannel
jne     DeleteAddressNextFilter

mov     [edi].FilterChannel, RBD_NOT_USED
mov     eax, [edi].FilterCmdBkIndex
mov     [ebp].EblkBkBusyFlags[ebx], 0

pushfd
cli
push    ecx
mov     ecx, size EblkStruct
mul     ecx
mov     edx, [ebp].IOMsgRamPtr
add     eax, 18100h
shr     eax, 1
add     eax, 3
push    eax
push    dx, ax
out     dx, ax
mov     edx, [ebp].IOMsgRam
xor     eax, eax
out     dx, ax
out     dx, ax
out     dx, ax

pop     eax
pop     ecx
sub     eax, 3
mov     edx, [ebp].IOMsgRamPtr
out     dx, ax
mov     edx, [ebp].IOMsgRam
mov     eax, 1
out     dx, ax
popfd

DeleteAddressNextFilter:
add     edi, size FilterStruct
dec     ch
jne     DeleteAddressFilterLoop

DeleteAddressNext:
add     ebx, 6
dec     cl
jne     DeleteAddressLoop

xor     eax, eax
ret

DeleteAddressError:
mov     eax, -1

```

```

DeleteAddress     endp
subttl    -- RegisterAgentSendRoutine --
page
;*****
; BEGIN_MANUAL_ENTRY( RegisterAgentSendRoutine, DPC/API/DELAADDR )
;
; Name:      RegisterAgentSendRoutine
;
; Description: This routine is called by DriverManagement to
;              register a Slip Send routine. The first dword in the first
;              ECB fragment points to the Slip Send routine to use. To
;              deregister the send routine, set the dword to a NULL.
;
; On Entry:   EAX    N/A
;              EBX    Frame Data Space
;              ECX    N/A
;              EDX    N/A
;              EBP    Adapter Data Space
;              ESI    ECB
;              EDI    N/A
;
; Note:       Interrupts are in any state.
;
; On Return:  EAX    Destroyed
;              EBX    Preserved
;              ECX    Destroyed
;              EDX    Destroyed
;              EBP    Preserved
;              ESI    Preserved
;              EDI    Preserved
;
; Flags:
;
; Note:       Interrupts preserved.
;
; Remarks:    This routine is called by DriverManagement.
;              It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;*****
; RegisterAgentSendRoutine proc
;
; mov     esi, [esi].RPacketOffset          ; ESI -> config structur
;
; mov     eax, [esi]
; EAX -> Slip Send Routine Address
; mov     [ebp].AgentSendRoutine, eax      ; Save it for later
; xor     eax, eax
; ret
;
; RegisterAgentSendRoutine     endp
subttl    -- RegisterAgent --
page
;*****
; BEGIN_MANUAL_ENTRY( RegisterAgent, DPC/API/REGAG )
;

```

; Name: RegisterAgent

; Description: This routine is called by DriverManagement to register package delivery/internet agent. The first dword in the first ECB fragment points to the Remove routine that we must call before we are removed.

; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A

; Note: Interrupts are in any state.

; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved

; Flags:

; Note: Interrupts preserved.

; Remarks: This routine is called by DriverManagement.
; It is called at process time.

; See Also:

; END_MANUAL_ENTRY

RegisterAgent proc

```
mov esi, [esi].RPacketOffset ; ESI -> config structur
mov eax, [esi]
EAX -> Slip Send Routine Address
xor [ebp].AgentRemoveRoutine, eax ; Save it for later
ret
```

RegisterAgent endp

```
subttl -- ReturnTCBCompleteRoutine --
page
```

; BEGIN_MANUAL_ENTRY(ReturnTCBCompleteRoutine, DPC/API/RETADS)

; Name: ReturnTCBCompleteRoutine

; Description: This routine is called by DriverManagement to return the TCB Complete routine. The first dword in the first ECB fragment points to a LONG which we will return with a pointer to the TCB complete routine.

; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A

; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A

; Note: Interrupts are in any state.

; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved

; Flags:

; Note: Interrupts preserved.

; Remarks: This routine is called by DriverManagement.
; It is called at process time.

; See Also:

; END_MANUAL_ENTRY

; if TIMESTAMP

```
extrn GetHighResolutionTimer: near
extrn HighResolutionTimer: dword
public DPCTimeStamp
DPCTimeStamp proc
```

; CPush

; call GetHighResolutionTimer

```
and eax, 00ffffffh
mov edx, timestamp_index
mov ecx, [esp + Parm0]
shl ecx, 24
or eax, ecx
mov [edx], eax
```

```
add edx, 4
cmp edx, timestamp_end
jae short DPCTimeStampBegin
mov timestamp_index, edx
mov dword ptr [edx], '$$$'
CPop
ret
```

DPCTimeStampBegin:

```
mov edx, timestamp_begin
jmp DPCTimeStampExit
```

DPCTimeStamp endp

endif

```
subttl -- DriverTCBComplete --
page
```

DriverTCBComplete proc

; CPush


```

; Name: RefreshMipsStats
; Description: This routine is called by to read the Mips stats
; from the adapter and to store them into the Local
; Mips Stats structure.
; On Entry: EAX N/A
; EBX N/A
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Destroyed
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by GetMipsStats and
; DriverCallBack.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; *****
RefreshMipsStats proc
    pushfd
    cli
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, 18700h / 2
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    lea     edi, [ebp].MipsRxEnables
    mov     ecx, (size StatsBlk) / 2
    cld
GetMipsStatsLoop:
    in     ax, dx
    stosw
    loop   GetMipsStatsLoop
    popfd
    xor     eax, eax
    ret
RefreshMipsStats     endp
subttl -- GetMipsStats --
page
; *****
; Description: This routine will modify the NIC's multicast registers to
; enable it to receive the multicast addresses listed in
; the multicast table. Each entry in the multicast table is as
; follows:

```

```

; BEGIN_MANUAL_ENTRY( GetMipsStats, DPC/API/GETMIPS )
; Name: GetMipsStats
; Description: This routine is called by DriverManagement to
; return the Mips statistics.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverManagement.
; It is called at process time.
; See Also:
; END_MANUAL_ENTRY
; *****
GetMipsStats proc
    push     esi
    call    RefreshMipsStats
    pop     esi
    mov     edi, [esi].RPacketOffset
    lea     esi, [ebp].MipsRxEnables
    mov     ecx, (size StatsBlk) / 4
    cld
    rep     movsd
    xor     eax, eax
    ret
GetMipsStats     endp
; *****
; BEGIN_MANUAL_ENTRY( DriverMulticastChange, DPC/API/MULTI )
; Name: DriverMulticastChange
; Description: This routine will modify the NIC's multicast registers to
; enable it to receive the multicast addresses listed in
; the multicast table. Each entry in the multicast table is as
; follows:

```